Distributed and Peer-to-Peer Data Mining for Scalable Analysis of Data from Virtual Observatories: Progress Report*

PI: Hillol Kargupta

August 22, 2007 - August 15, 2008

1 Introduction

Design, implementation, and archival of very large sky surveys are playing an increasingly important role in todays astronomy research. Current projects such as GALEX All-Sky Survey and future ones such as WISE All-Sky Survey are destined to produce enormous catalogs of astronomical sources. The Large Synoptic Survey Telescope is supposed to stream in large volumes of data at a high rate. It is this virtual collection of gigabyte, terabyte, and (eventually) petabyte catalogs and streams that will enable remarkable new scientific discoveries through the integration and cross-correlation of data across these multiple survey dimensions. However, this will be difficult to achieve without a computational backbone that includes support for queries and data mining across distributed virtual tables of de-centralized, joined, and integrated sky survey catalogs. Moreover, use of local data management systems such as MyDB, MySpace in AstroGrid, and Grid Bricks for storing and managing user's local data is becoming increasingly popular. This is opening up the possibility of constructing Peer-to-Peer (P2P) networks for data sharing and mining.

This research is exploring the possibility of using distributed and P2P data mining technology for exploratory astronomy from data integrated and cross-correlated across these multiple sky surveys. It is considering several scientific problems in order to illustrate the possibilities. For example, we are exploring classical fundamental plane problem in a new light which is trying to answer some of the following questions: How does local galactic density relate to galactic fundamental plane structure? Does the fundamental plane structure of galaxies in low density regions differ from that of galaxies in high density regions? Since the attributes which define the fundamental plane span two data repositories SDSS and 2MASS instead of one, we focus on cross-matching them available individually through the NVO. We are using distributed data mining algorithms to analyze this data distributed over a large number of nodes.

The National Virtual Observatory (NVO) was developed in an effort to address this problem. The NVO provides web-based services allowing the public to download data from many different, autonomous sky surveys. In this project, we are examining the use of distributed data mining technology on top of the NVO as a tool for allowing the Astronomers to tap the riches of data integrated and cross-correlated across these multiple sky surveys. We are examining two modes of inquiry. First, with large amounts of data downloaded and cross-matched between multiple catalogs (a difficult and time-consuming task), apply data mining technology to address astrophysical questions that could not be as readily addressed using only individual catalogs (see Section 2) and support for standard data-query processing. Second, develop communication efficient, distributed data mining techniques allowing those types of analysis to be performed across multiple catalogs without first downloading them to a central location (see Section 3). We discuss the future work in Section 5.

2 Data Mining on the NVO

The goal of this part of the research is to explore a few important types of data mining applications that are likely to impact the field of astronomy significantly. We would like to first identify the possibilities and

*Award NNX07AV70G

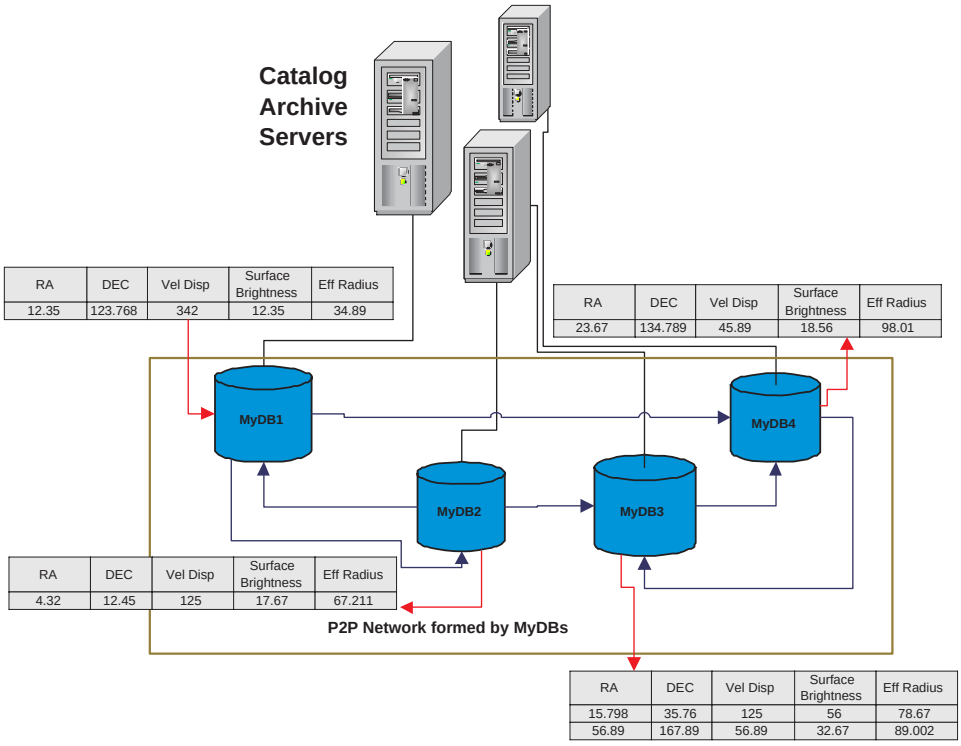


Figure 1: P2P Data Mining on Homogeneously Partitioned Sky Survey.

then develop distributed data mining technology-based scalable, communication-efficient solutions.

2.1 DEMAC - A System for Distributed Exploration of Massive Astronomic Catalogs

This section describes the high level design of the DEMAC system. DEMAC is designed as an additional Web-Service which seamlessly integrates into the NVO. It consists of two basic services. The main one is Web-Service which provides Distributed and Peer-to-Peer Data Mining capabilities for sky surveys (WS-DPDM). The second one, which is intensively used by WS-DPDM is a Web-Service which provides Cross-Matching capabilities for vertically distributed sky surveys (WS-CM). Cross-matching of sky surveys is a complex topic which is dealt with, in itself, under other NASA funded projects. Thus, our implementation of this Web-Service would supply bare minimum capabilities which are required in order to provide distributed data mining capabilities.

This Web-Service will allow running distributed data mining algorithms on a selection of sky-surveys. The user would use existing NVO services to locate sky-surveys and define the portion of the sky to be mined. This process could include: (1) individual catalogs or (2) virtual tables. If the choice is to mine a single catalog, the user may be interested in contacting his nearby peers to obtain subsets of the archived catalog. In such a scenario, the WS-DPDM service will facilitate routing of queries to other nearby peers on the network. On the other hand, if a virtual catalog is required, the user would need to make use of WS-CM to select a cross-matching scheme for those sky-surveys. This specifies how the tuples are matched across surveys to define the virtual table to be analyzed. Following these two preliminary phases the user would submit the data mining task. Figure 1 shows a schematic diagram of the DEMAC architecture for P2P astronomy data mining.

2.2 Data Sets

We have focused on the SDSS and 2MASS repositories available individually through the NVO. We have created a large, hybrid dataset by downloading the 2MASS XCS catalog and cross-matched it against the SDSS catalog (using the SDSS Crossid tool).[1] We filtered the data based on the SDSS identified type to remove all non-galaxy tuples. At this point, all tuples retained represent galaxies. We then filtered the data again based on reasonable redshift (actual or estimated) values ($0.003 <= z <= 0.300$). Finally, we computed a location for each tuple (galaxy) in 3D Euclidean space and used the Delaunay Tessellation Field Estimator to compute, for each tuple (galaxy), a measure of its local galactic density.

2.3 Mining the Relationship Between Local Galactic Density and Galactic Fundamental Plane Structure

At this point we had a 156,000 tuple (galaxy) dataset involving attributes from both the 2MASS and SDSS repository and, associated with each tuple, a measure of its local galactic density. This dataset, obtained with substantial effort, allows us to apply data mining technology to address astrophysical questions that could not be as readily addressed using only the SDSS or 2MASS catalog individually.

We are currently focusing on addressing the question of how local galactic density relates to galactic fundamental plane structure. Does the fundamental plane structure of galaxies in low density regions differ from that of galaxies in high density regions? We constructed two attributes using only attributes from SDSS: log(Petrosian I band angular effective radius*redshift) and log(velocity dispersion). Using these two and a third from 2MASS, K band mean surface brightness, we had a dataset allowing us to address the question more readily than we could have using only data from SDSS or 2MASS alone. We are submitting a summary of our findings to the AGU Fall meeting 2008 for consideration by astronomers for significance. We envision this study as only the beginning of a longer series of studies designed to address astrophysical questions that could not be as readily addressed using only the SDSS or 2MASS catalog individually.

Using equi-depth bins with respect to the local galactic density, we partitioned the above, 3 attribute dataset into 30 parts. For each part, we carried out the fundamental plane computation (*i.e.* PCA) and the results are reported in Figure 2.3.

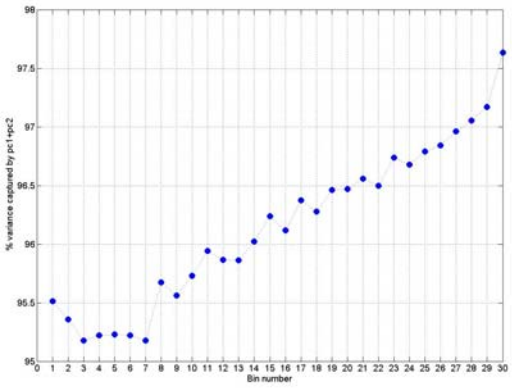
2.4 Astrophysical Significance

Astronomers discovered an interesting correlation among galaxy properties nearly 20 years ago, which led to a greater understanding of the structure and formation of these fundamental building blocks of our Universe. That correlation demonstrated that 3 of the major measured parameters for elliptical (spheroidal) galaxies are reduced to a plane in 3-d parameter space. This has come to be known as the fundamental plane of elliptical galaxies. Subsequently, it was demonstrated that this behavior is a result of simple gravitational physics. The 3 parameters are size (radius), density (surface brightness), and internal velocity (velocity dispersion). These 3 parameters are related by a physical relationship, known as the Virial Theorem, thus reducing the dimensionality of the measured parameter set to two independent variables. A few years before the discovery of the fundamental plane, astronomers had already discovered and began investigating intensely another relationship: the morphology-density relation. This relationship revealed that the morphological type distribution of galaxies (such as ellipticals versus spiral galaxies) depends strongly on the local galaxy environment (the local density of galaxies). In particular, the higher the density, then the higher the ratio of number of ellipticals to number of spirals, and vice versa.

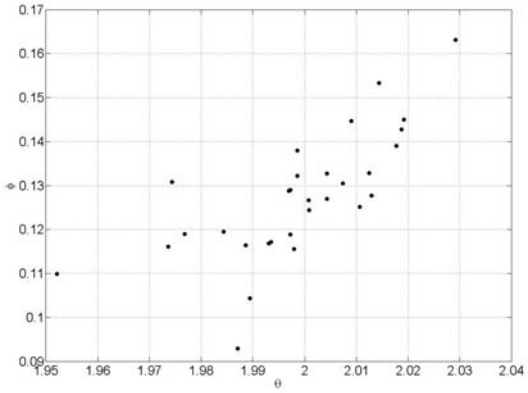
With the development of major astronomical sky surveys, it is now possible to test these relationships and their inter-relation in greater detail than ever before possible. Specifically, we have used a combination of two major sky surveys (Sloan Digital Sky Survey [SDSS], and the 2-Micron All-Sky Survey [2MASS]) to test numerous astrophysical hypotheses.

The first and most obvious hypothesis that we tested was to re-discover the fundamental plane of elliptical galaxies using a small subset of the SDSS and 2MASS galaxy catalogs. In recognition of the fact that we are doing research into efficient algorithms for distributed data mining (DDM; i.e., distributed mining of

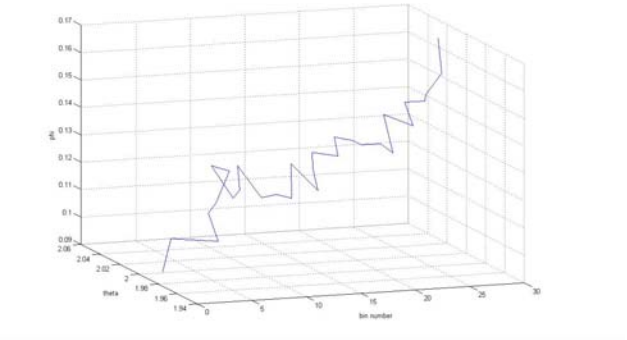
[1] All attributes of the *PhotoObjAll* and *SpecObjAll* tables as well as an estimated redshift attribute from the *Photoz2* table were obtained and joined to the cross-matched tuples from the 2MASS XCS catalog.



(a) Variance captured by first two principal components for each of the 30 bins. Bin 1 has the lowest mean density and Bin 30 the highest.



(b) Scatter plot of θ versus ϕ. For each bin, the unit normal vector to the first two principal components is computed and converted to spherical polar coordinates $(1\theta, \phi)$.



(c) Variation of θ and ϕ with bin number. The bins are numbered in increasing order of density.

distributed data), we initially focused on the DDM algorithm development, validating their outputs by demonstrating that we could recover the fundamental plane of elliptical galaxies with the same degree of accuracy as astronomers have done with smaller non-distributed data sets. We succeeded in these research activities.

The next step, as proposed in our NASA AISR project, was to take these information science algorithms and to apply them to NASA science applications on larger data collections. Again, in the interest of continuity of our research program, we focused on the galaxy relationships through mining of the SDSS and 2MASS catalogs. Our AISR program is aimed at developing and applying data mining on a large-scale, through distributed computing, on large data sets. So, we decided to start with a computational problem that is a good fit for the distributed data mining technology – testing and measuring the fundamental plane in a large number of different galaxy environments. In other words, we would partition the data sets into a large number of astrophysically independent sets and run the algorithms on these smaller sets. These independent subsets are partitioned by local galaxy environment density. Note that each of these "smaller sets" is in fact significantly larger than any set of galaxies previously analyzed in this way. Following this procedure, we discovered that the fundamental plane of elliptical galaxies does hold up across all galaxy environments. Plus, we made a new scientific discovery – that the strength of the correlation increases with local galaxy density. The higher the density, then the more variance is captured in the first PCA principal components. In addition, the direction of the fundamental plane (i.e., its normal vector) was found to change in a systematic manner as a function of density.

These new astrophysics discoveries are now being written up in a scientific paper to be published in a peer-reviewed astronomy journal. We will continue to investigate additional correlations for astrophysical significance, and then proceed to other NASA science disciplines to further test and improve our algorithms for distributed and peer-to-peer scientific data mining.

3 Distributed Data Mining on the NVO

We are developing a system for the distributed data mining over large astronomy catalogs. The system is to be built on top of the existing NVO and provides tools for data mining (as web services) without requiring datasets to be down-loaded to a centralized server. We envision the system as serving the role of an exploratory "browser". Users will be able to quickly get results for their distributed data analysis-queries at low communication cost. Armed with these results, users should be able to focus in on a specific query or portion of the datasets, and down-load for more intricate analysis, if necessary. This system will need distributed data mining algorithms. We have spent significant amount of time developing new communication-efficient distributed algorithms for PCA and outlier detection among others. The following part of this section discusses some of the algorithms developed during this research period.

3.1 Communication Efficient Distributed Probabilistic Algorithm for Mining Frequent Itemsets

This part of the research focuses on developing a communication efficient algorithm for discovering frequent itemsets from data distributed homogeneously in a networked environment. The classic frequent itemset mining algorithm does not work in this setting as it needs to access the data in its entirety available at one location. In a large-scale distributed environment, finding all the network-wide frequent itemsets with exact frequency is computationally difficult and usually requires large amount of communication (often communication per peer reaches order of the network size). Instead, this research tries to find most of the frequent itemsets with probabilistic guarantee and their approximate frequency using only a bounded communication. The research takes sampling based approach to identify frequent itemsets from the entire network. It first addresses the challenging problem of collecting an unbiased uniform sample from a network, and shows how to take a uniform sample of nodes and a uniform sample of data from a network using random-walk. It then applies the proposed sampling technique to identify most of the frequent itemsets from a network. Theoretical analysis shows how to bound the sample-size irrespective of the size of the entire data and hence, communication necessary to compute the results. Experimental analysis shows that the proposed algorithm discovers all of the network-wide frequent itemsets using only a bounded communication.

3.2 Approximate Distributed K-means Clustering

This part of the research offers the distributed K-means clustering problem where the data and computing resources are distributed over a large network. It offers two algorithms which produce an approximation of the result produced by the standard centralized K-means clustering algorithm. The first is designed to operate in a dynamic network that can produce clustering by "local" synchronization only. The second algorithm uses uniformly sampled peers and provides analytical guarantees regarding the accuracy of clustering on a network. Empirical results show that both the algorithms demonstrate good performance compared to their centralized counterparts at little communication cost.

3.3 A Generic Local Algorithm for Mining Data Streams in Large Distributed Systems

In a large network of computers or wireless sensors, each of the components (henceforth, peers) has some data about the global state of the system. Much of the systems functionality such as message routing, information retrieval and load sharing relies on modeling the global state. We refer to the outcome of the function (e.g., the load experienced by each peer) as the model of the system. Since the state of the system is constantly changing, it is necessary to keep the models up-to-date. Computing global data mining models e.g. decision trees, k-means clustering in large distributed systems may be very costly due to the scale of the system and due to communication cost, which may be high. The cost further increases in a dynamic scenario when the data changes rapidly. In this research, we describe a two step approach for dealing with these costs. First, we describe a highly efficient local algorithm which can be used to monitor a wide class of data mining models. Then, we use this algorithm as a feedback loop for the monitoring of complex functions of the data such as its k-means clustering. The theoretical claims are corroborated with a thorough experimental analysis

3.4 A Scalable Local Algorithm for Distributed Multivariate Regression

This research offers a local distributed algorithm for multivariate regression in large peer-to-peer environments. The algorithm can be used for distributed inferencing, data compaction, data modeling and classification tasks in many emerging peer-to-peer applications for bioinformatics, astronomy, social networking, sensor networks and web mining. Computing a global regression model from data available at the different peer-nodes using a traditional centralized algorithm for regression can be very costly and impractical because of the large number of data sources, the asynchronous nature of the peer-to-peer networks, and dynamic nature of the data/network. This research proposes a two-step approach to deal with this problem. First, it offers an efficient local distributed algorithm that monitors the quality of the current regression model. If the model is outdated, it uses this algorithm as a feedback mechanism for rebuilding the model. The local nature of the monitoring algorithm guarantees low monitoring cost. Experimental results observed in this component of the research strongly support the theoretical claims.

3.5 Distributed Decision Tree Induction in Peer-to-Peer Systems

This part of the research offers a scalable and robust distributed algorithm for decision tree induction in large Peer-to-Peer (P2P) environments. Computing a decision tree in such large distributed systems using standard centralized algorithms can be very communication-expensive and impractical because of the synchronization requirements. The problem becomes even more challenging in the distributed stream monitoring scenario where the decision tree needs to be updated in response to changes in the data distribution. This research presents an alternate solution that works in a completely asynchronous manner in distributed environments and offers low communication overhead, a necessity for scalability. It also seamlessly handles changes in data and peer failures. This research presents extensive experimental results to corroborate the theoretical claims.

3.6 Local Algorithms for Distributed Multivariate Regression in Peer-to-Peer Networks

This research offers a local distributed algorithm for multivariate regression in large peer-to-peer environments. The algorithm is designed for distributed inferencing, data compaction, data modeling and classification tasks in many emerging peer-to-peer applications for bioinformatics, astronomy, social networking, sensor networks and web mining. Computing a global regression model from data available at the different peer-nodes using a traditional centralized algorithm for regression can be very costly and impractical because of the large number of data sources, the asynchronous nature of the peer-to-peer networks, and dynamic nature of the data/network. This research proposes a two-step approach to deal with this problem. First, it offers an efficient local distributed algorithm that monitors the quality of the current regression model. If the model is outdated, it uses this algorithm as a feedback mechanism for rebuilding the model. The local nature of the monitoring algorithm guarantees low monitoring cost. Experimental results strongly support the theoretical claims.

3.7 Distributed Identification of Top-l Inner Product Elements and its Application in a Peer-to-Peer Network

Inner product measures how closely two feature vectors are related. It is an important primitive for many popular data mining tasks, e.g., clustering, classification, correlation computation, and decision tree construction. If the entire data set is available at a single site, then computing the inner product matrix and identifying the top (in terms of magnitude) entries is trivial. However, in many real-world scenarios, data is distributed across many locations and transmitting the data to a central server would be quite communication-intensive and not scalable. This research offers an approximate local algorithm for identifying top-l inner products among pairs of feature vectors in a large asynchronous distributed environment such as a peer-to-peer (P2P) network. We develop a probabilistic algorithm for this purpose using order statistics and Hoeffding bound. We present experimental results to show the effectiveness and scalability of the algorithm. Finally, we demonstrate an application of this technique for interest-based community formation in a P2P environment.

3.8 Distributed Top-K Outlier Detection from Astronomy Catalogs using the DEMAC System

The design, implementation and archiving of large sky surveys is an important part of astronomy research. The Sloan Digital Sky Survey (SDSS), The Two Micron All Sky Survey (2MASS) are some such surveys producing tera bytes of geographically distributed data which need to be stored, analyzed and queried to enable scientific discoveries. In this research, we develop the architecture of a system for Distributed Exploration of Massive Astronomy Catalogs (DEMAC) which is built on top of the existing National Virtual Observatory environment. We describe distributed algorithms for doing Principal Component Analysis (PCA) using random projection and sampling based techniques. Using the approximate principal components, we develop a distributed outlier detection algorithm which enables identification of data points that deviate sharply from the "correlation structure" of the data. We provide simulation results with data obtained from sky-surveys SDSS and 2MASS.

4 Milestones for Year I: Review

This section describes the achieved milestones for the first year.

1. Detailed design of a research prototype based on various web-based distributed data mining algorithms.
2. Establishment of local testbed including 3-4 datasets.
3. Modeling of the astronomy research problem, development of user requirements and additional algorithms required, detailed design of a catalog alignment Web-Service.

The astronomical problem we have chosen is the fundamental plane problem and we have decided to identify galatic properties that uniquely determine the fundamental problem. We have chosen our data sources as the SDSS and 2MASS databases. In the current test-bed, we have used cross-matched data from these separate sources. The results have shown that galactic density can be considered an important galactic property that determines the fundamental plane. Simultaneously, we have developed distributed algorithms for distributed PCA computation, distributed outlier detection and distributed ordering using an order statistics based distributed algorithm. These algorithms plays a key role in the design of the proposed distributed data mining system for virtual observatories.

5 Future Work

The immediate goals for the second year research are as follows:

1. Development of additional distributed and peer-to-peer distributed data mining algorithms.
2. Implementation of the prototype system along with the distributed data mining algorithms.
3. Testing of the prototype using the identified astronomy problem.
4. Web-based user interface development.

Publications

Peer Reviewed Publications

1. H. Dutta, C. Giannella, K. Borne, H. Kargupta, and R. Wolff. (2008) Distributed Data Mining for Astronomy Catalogs. Submitted to the IEEE Transactions on Knowledge and Data Engineering. (In communication).
2. K. Das, W. Griffin, H. Kargupta, C. Giannella, K. Borne. (2008). Scalable Multi-Source Astronomy Data Mining in Distributed, Peer-to-Peer Environments. Abstract submitted to 18th annual ADASS conference.
3. S. Datta and H. Kargupta.(2008). A Communication Efficient Probabilistic Algorithm for Mining Frequent Itemset from Peer-to-Peer Network. Statistical Analysis and Data Mining Journal. (In Press).
4. S. Datta, C. Giannella and H. Kargupta. (2008). Approximate K-means Clustering Over a Peer-to-peer Network. IEEE Transactions on Knowledge and Data Engineering. (In communication)
5. R. Wolff, K. Bhaduri, H. Kargupta. (2008). A Generic Local Algorithm for Mining Data Streams in Large Distributed Systems. IEEE Transactions on Knowledge and Data Engineering. (Accepted, in press).
6. K. Bhaduri, H. Kargupta. (2008). A Scalable Local Algorithm for Distributed Multivariate Regression. Statistical Analysis and Data Mining Journal (Accepted, in press).
7. K. Bhaduri, R. Wolff, C. Giannella, H. Kargupta. (2008). Distributed Decision Tree Induction in Peer-to-Peer Systems. Statistical Analysis and Data Mining. Volume 1, Issue 2, pp. 85-103.
8. K. Bhaduri, H. Kargupta. (2008). An Efficient Local Algorithm for Distributed Multivariate Regression in Peer-to-Peer Networks. SIAM International Conference on Data Mining, Atlanta, Georgia. pp. 153-164. (Best of SDM'08).
9. K. Das, K Bhaduri, K. Liu, and H. Kargupta. Distributed Identification of Top-l Inner Product Elements and its Application in a Peer-to-Peer Network. IEEE Transactions on Knowledge and Data Engineering (TKDE), Vol. 20, No. 4, pp. 475-488, April 2008.

10. H. Kargupta. (2007). Thoughts on Human Emotions, Communication Breakthroughs, and the Next Generation of Data Mining. National Science Foundation Symposium on Next Generation of Data Mining and Cyber-Enabled Discovery for Innovation.

11. S. Datta, H. Kargupta. (2007). Uniform Data Sampling from a Peer-to-Peer Network. Proceedings of the 2007 IEEE International Conference on Distributed Computing Systems (ICDCS 2007), Toronto, Canada, June, 2007, pp. 50.

12. H. Dutta, C. Giannella, K. Borne and H. Kargupta. (2007). Distributed Top-K Outlier Detection from Astronomy Catalogs using the DEMAC System. Proceedings of the SIAM International Conference on Data Mining, Minneapolis, USA, April 2007.

13. S. Mukherjee, H. Kargupta. (2007). Distributed Probabilistic Inferencing in Sensor Networks using Variational Approximation. Journal of Parallel and Distributed Computing (JPDC), Volume 68, Issue 1, January 2008, Pages 78-92.

Dissertations

1. Haimonti Dutta, *Empowering Scientific Discovery by Distributed Data Mining on the Grid Infrastructure*, Ph.D. Dissertation, University of Maryland, Baltimore County. 2007.

2. Kanishka K. Bhaduri. Efficient Local Algorithms for Distributed Data Mining in Large Scale Peer to Peer Environments: A Deterministic Approach. PhD Dissertations. University of Maryland, Baltimore County. 2008.

3. Approximate Distributed Algorithms for Mining Data in Peer-to-Peer Networks. PhD Dissertations. University of Maryland, Baltimore County. 2008.

Appendix I: Sample Publications

Distributed Data Mining for Astronomy Catalogs

Haimonti Dutta*, Chris Giannella†, Kirk Borne§, Hillol Kargupta*‡, and Ran Wolff¶

*Department of Computer Science and Electrical Engineering,
University of Maryland Baltimore County, Baltimore, MD 21250
Email:{hdutta1,hillol}@csee.umbc.edu

†Department of Computer Science,
Loyola College, Baltimore, MD 21210
Email:{cgiannel@acm.org}

§Department of Computational and Data Sciences,
George Mason University, Fairfax, VA 22030
Email: {kborne@gmu.edu}

¶Management Information Systems,
Haifa University, Israel.
Email:{rwolff@mis.haifa.ac.il}

‡The author is also affiliated to Agnik, LLC., Columbia, MD.

A 6-page version of this paper was published in the Proceedings of the 2007 SIAM International Conference on Data Mining.

Abstract

The design, implementation and archiving of very large sky surveys is playing an increasingly important role in today's astronomy research. These data archives are geographically distributed and heterogeneous in nature. Existing techniques for knowledge extraction from this data require downloading the archives to a centralized site. However, merging of remote data at a central site to perform data mining will result in unnecessary communication overhead. We believe, to fully exploit the potential of this data, mechanisms ought to be provided for more communication efficient multiple archive data analysis.

In this paper, we describe a system for Distributed Exploration of Massive Astronomical Catalogs (DEMAC). The system is designed to be integrated on top of the existing National Virtual Observatory environment and provides tools for distributed data mining (as web services) without requiring datasets to be fully down-loaded to a centralized server. To illustrate the potential effectiveness of our system, we develop communication-efficient distributed algorithms for (1) Principal Component Analysis (PCA) and (2) outlier detection. We perform case studies on real Astronomy data to evaluate the performance of our algorithms on the fundamental plane of astronomical parameters. In particular, PCA enables dimensionality reduction within a set of correlated physical parameters such as a reduction of a 3-dimensional data distribution (in astronomer's observed units) to a planar data distribution (in fundamental physical units). Outlier detection enables identification of "interesting" galaxies that do not fall in this planar distribution. Fundamental physical insights are thereby enabled through efficient access to distributed multi-dimensional data sets.

Index Terms

Distributed Data Mining, Astronomy Catalogs, Cross Matching, Principal Component Analysis, Outlier Detection.

I. INTRODUCTION

The design, implementation and archiving of very large sky surveys is playing an increasingly important role in today's astronomy research. Many projects (*e.g.* GALEX All-Sky Survey, WISE All-Sky Survey and LSST Large Synoptic Survey) are producing enormous catalogs (tables) of astronomical sources (tuples). These catalogs are geographically distributed. If science progressed through these individual data archives alone, then there is no problem. However, some of the greatest scientific discoveries have come at the intersection of different disciplines – in astronomy, this means at the intersection of different wavelength domains. For example: the most luminous

galaxies in the Universe (Ultra-Luminous and Hyper-Luminous IR Galaxies) were found through the comparison of IR and optical data; Quasars and Powerful Radio Galaxies were found through the combined analysis of radio and optical data. To make great progress in astronomy, it is imperative to link these distributed data collections and to provide tools that enable analysis of distributed data.

In this paper, we describe a system for the Distributed Exploration of Massive Astronomical Catalogs (DEMAC). DEMAC offers a collection of data mining tools based on various DDM algorithms. The system is built on top of the existing National Virtual Observatory [1] environment and provides tools for data mining (as web services) without requiring datasets to be down-loaded to a centralized server. The algorithms we develop sacrifice accuracy for communication savings. They offer approximate results at a considerably lower communication cost than that of exact results through centralization. As such, we see DEMAC as serving the role of an exploratory "browser". Users can quickly get (generally quite accurate) results for their distributed queries at low communication cost. Armed with these results, users can focus in on a specific query or portion of the datasets, and down-load for more intricate analysis.

Since these attributes are now necessarily distributed across geographically dispersed data archives, it is scientifically valuable to explore distributed Principal Component Analysis (PCA) and outlier detection on larger astronomical data collections and for greater numbers of astrophysical parameters. The application of communication-efficient distributed PCA and outlier detection along with other DDM algorithms will likely enable new scientific insights into our Universe.

To illustrate the potential effectiveness of our system, we develop communication-efficient distributed algorithms for PCA and outlier detection. We carry out case studies using distributed PCA for detecting fundamental planes of astronomical parameters. Astronomers have previously discovered cases where the observed parameters measured for a particular class of astronomical objects (such as elliptical galaxies) are strongly correlated, as a result of universal astrophysical processes (such as gravity). PCA will find such correlations in the form of principal components.

Examination of a subset of the parameter space can also help astronomers identify objects with atypical behavior ([2],[3]). It is therefore important to systematically explore the observable parameter space, and specifically search for rare, unusual, or previously unknown types of astronomical objects and phenomena. For example, examination of the three dimensional parameter

space formed by the difference of petrosian flux in the R and I bands, the K-band mean surface brightness and logarithm of the K-band concentration index may lead to identification of galaxies that were previously not known to be “interesting". This motivates the need for designing outlier detection algorithms.

The rest of the paper is organized as follows. Section II discusses related work on analysis of large scientific data collections as well as DDM. Section III describes the architecture of the DEMAC system. Section IV presents the data analysis problem addressed in this paper: analyzing distributed astronomical virtual catalogs. Section V reviews PCA background material. Sections VI and VII describe distributed algorithms for virtual catalog PCA and outlier detection, respectively. Section VIII provides a two case studies to evaluate the effectiveness of our distributed techniques: finding galactic fundamental planes and galactic outlier detection. Finally, Section IX concludes the paper.

II. Related Work

A. Analysis of Large Scientific Data Collections

There are several instances in the astronomy and space sciences research communities where data mining is being applied to large data collections ([4], [5]). Some dedicated data mining projects include F-MASS [6], Class-X [7], the Auton Astrostatistics Project [8], and additional VO-related data mining activities (such as SDMIV [9]). In essentially none of these cases does the project involve truly DDM [10], [11], [12].

One of the first large-scale attempts at grid data mining for astronomy is the U.S. National Science Foundation (NSF) funded GRIST [13] project. The GRIST goals include application of grid computing and web services (service-oriented architectures) to mining large distributed data collections. GRIST is focused on one particular data modality: images. Hence, GRIST aims to deliver mining on the pixel planes within multiple distributed astronomical image collections. The project that we are proposing here is aimed at another data modality: catalogs (tables) of astronomical source attributes. GRIST and other projects also strive for exact results, which usually requires data centralization and co-location, which further requires significant computational and communications resources. DEMAC (our system) produces approximate results without requiring data centralization (low communication overhead). Users can quickly get (generally quite accurate) results for their distributed queries at low communication cost. Armed with these

results, users focus in on a specific query or portion of the datasets, and down-load for more intricate analysis.

The U.S. National Virtual Observatory (NVO) [1] is a large scale effort funded by the NSF to develop a information technology infrastructure enabling easy and robust access to distributed astronomical archives. It will provide services for users to search and gather data across multiple archives and some basic statistical analysis and visualization functions. It will also provide a framework for new services to be made available by outside parties. These services can provide, among other things, specialized data analysis capabilities. As such, DEMAC fits nicely into the NVO as a new service.

The International Virtual Observatory (IVO) [14] is another large scale effort to develop an infrastructure enabling easy and robust access to distributed astronomical archives. Generally, the Virtual Observatory can be seen as part of an ongoing trend toward the integration of information sources. The main paradigm used today for the integration of these data systems is that of a data grid [15], [16], [17], [18], [19], [20], [21], [22]. Among the desired functionalities of a data grid, data analysis takes a central place. As such, there are several projects [23], [24], [25], [13], [26], [27], which in the last few years attempt to create a data mining grid. In addition, grid data mining has been the focus of several workshops [28], [29].

B. Distributed Data Mining

DDM is a relatively new technology that has been enjoying considerable interest in the recent past [30], [11]. DDM algorithms strive to analyze the data in a distributed manner without down-loading all of the data to a single site (which is usually necessary for a regular centralized data mining system). DDM algorithm naturally fall into two categories according to whether the data is distributed horizontally (with each site having some of the tuples) or vertically (with each site having some of the attributes for all tuples). In the latter case, it is assumed that the sites have an associated unique id[1] used for matching. In other words, consider a tuple t and assume site A has a part of this tuple, t_A, and B has the remaining part t_B. Then, the id associated with t_A equals the id associated with t_B.[2]

[1]The unique id can be a primary key.

[2]Each id is unique to the site at which it resides; But, ids can match across sites; a tuple at site A can have the same id as a tuple at site B.

The NVO can be seen as a case of vertically partitioned data, assuming ids have been generated by a cross-matching service. With this assumption, DDM algorithms for vertically partitioned data can be applied. These include algorithms for PCA [31], [32], clustering [33], [31], Bayesian network learning [34], [35], and supervised classification [36], [37], [38], [39], [40]. Kargupta and Puttagunta [32] developed a randomized algorithm for distributed Principal Component Analysis from vertically partitioned data. Our work in [41] is a slightly revised version of [32].

C. Outlier Detection

Due to the fact that the definition of an outlier is inherently imprecise, many approaches have been developed for detecting outliers. We will not attempt a comprehensive citation listing, instead, the reader is referred to Hodge and Austin [42] for an excellent survey mostly focusing on outlier detection methodologies from machine learning, pattern recognition, and data mining. Also, Barnett and Lewis [43] provide and excellent survey of outlier detection methodologies from statistics. Note, some work has been done on parallel algorithms for outlier detection on *horizontally* distributed data (not in an astronomy context) [44]. However, to our knowledge, no work has been done on outlier detection over vertically distributed data.

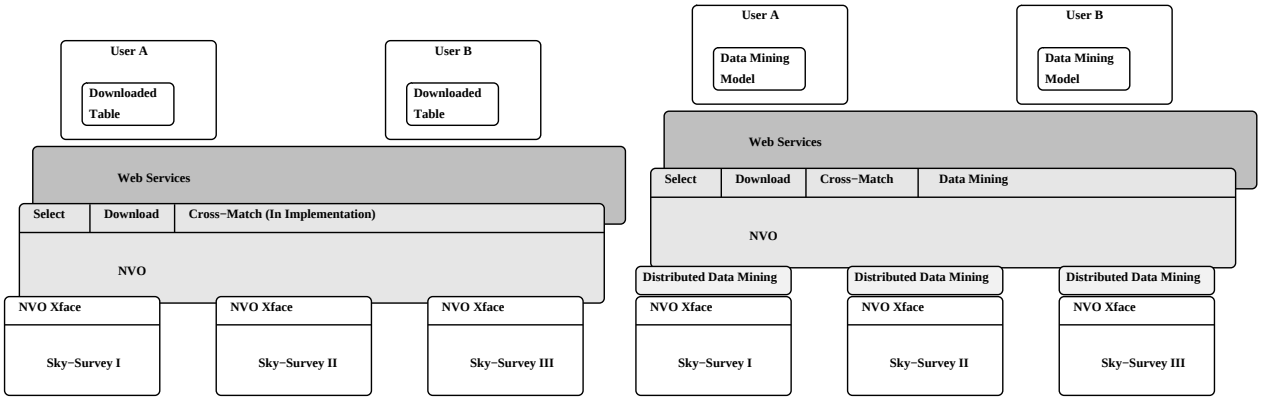
D. Our Prior Work

This paper extends our prior work in [41] and [45] in the following ways.

1) We have greatly extended the discussion of the system architecture and related work.
2) For distributed outlier detection algorithm, we have modified the algorithm to require only one invocation to find the top k outliers rather than k separate invocations. Moreover, we have extended the correctness proof to include all its details.
3) For the distributed outlier detection experiments, we have replaced the accuracy metric with two more meaningful ones and have carried out more extensive experiments. More comments on this are provided in Section VIII.

III. DEMAC - A System for Distributed Exploration of Massive Astronomical Catalogs

In order to meet the challenges of the multi-archive data analysis problem faced by the astronomy community, the U.S. National Virtual Observatory(NVO) [1] and an International



(a) Current data flow is restricted because of data ownership and bandwidth considerations.

(b) Distributed data mining algorithms can process large amounts of data using a small amount of communication. The users get the data mining output rather than raw data.

Fig. 1. Proposed data flow for distributed data mining embedded in the NVO.

Virtual Observatory (IVO) [14] has been developed and deployed. However, processing, mining, and analyzing these distributed and vast data collections are fundamentally challenging tasks since most off-the-shelf data mining systems require the data to be down-loaded to a single location before further analysis. This imposes serious scalability constraints on the data mining system and fundamentally hinders the scientific discovery process. Figure 1 further illustrates this technical problem. The left part depicts the current data flow in the NVO. Through web services, data are selected and down-loaded from multiple sky-surveys.

Our system requires a conceptually simple modification – the addition of a distributed data mining functionality in the sky servers. This allows DDM to be carried out without having to down-load large tables to the users' desktop or some other remote machine. Instead, the users will only down-load the output of the data mining process (a data mining model); the actual data mining from multiple data servers is performed using communication-efficient DDM algorithms.

This section describes the high level design of the DEMAC system. DEMAC is designed as an additional web-service which seamlessly integrates into the NVO. It consists of two basic services. The main one is a web-service providing DDM capabilities for vertically distributed sky surveys (*WS-DDM*). The second one, which is intensively used by WS-DDM, is a web-service providing cross-matching capabilities for vertically distributed sky surveys (*WS-CM*).

To provide a distributed data mining service, DEMAC relies on other services of the NVO

such as the ability to select and down-load from a sky survey using SQL. Key to our approach is that these services be used not over the web, through the NVO, but rather by local agents which are co-located with the respective sky survey. In this way, the DDM service avoid bandwidth and storage bottlenecks, and overcomes restrictions which are due to data ownership concerns. Agents, in turn, take part in executing efficient distributed data mining algorithms, which are highly communication-efficient. It is the outcome of the data mining algorithm, rather than the selected data table, that is provided to the end-user. With the removal of the network bandwidth bottleneck, the main factor limiting the scalability of the distributed data mining service would be database access. For database access we intend to rely on the SQL interface provided by the different sky-surveys to the NVO.

We outline here the architecture for the two web-services in the DEMAC system.

A. WS-DDM – DDM for Heterogeneously Distributed Sky-Surveys

This web-service allows running a DDM algorithm on a selection of sky-surveys. The user uses existing NVO services to locate sky-surveys and define the portion of the sky to be data mined. Then the WS-CM is used to select a cross-matching scheme for those sky-surveys. This specifies how the tuples are matched across surveys to define the virtual table to be analyzed[3]. Following these two preliminary phases the user submits the data mining task.

Execution of the data mining task is scheduled according to resource availability. Specifically, the size of the virtual table selected by the user dictates scheduling. Having allocated the required resources, the data mining algorithm is run by agents which are co-located with the selected sky-surveys. These agents access the sky-survey through the SQL interface it exposes to the NVO and communicate with each other directly, over the Internet. When the algorithm has terminated, results are provided to the user using a web-interface.

B. WS-CM – Cross-Matching for Heterogeneously Distributed Sky-Surveys

Before the application of any distributed data mining algorithm on the astronomy catalogs, we must first consider the problem of cross-matching the catalogs. The process can be briefly

[3]The procedure for cross-matching is illustrated by an example in section III-B

ID	RA	DEC	A
P_1	ra_1	dec_1	a_1
P_2	ra_2	dec_2	a_2
P_3	ra_3	dec_3	a_3

ID	RA	DEC	B
Q_1	ra_4	dec_4	b_1
Q_2	ra_2	dec_2	b_2
Q_3	ra_1	dec_1	b_4

TABLE I

CATALOGS P AND Q.

RA	DEC	A	B
ra_1	dec_1	a_1	b_4
ra_2	dec_2	a_2	b_2

TABLE II

VIRTUAL CATALOG BETWEEN P AND Q.

illustrated as follows:

1) Consider the catalogs P and Q shown in Table I. Each record corresponds to a celestial object and contains right ascension (RA) and declination (DEC) attributes indicating the position of the object in the celestial sphere. The records in catalog P have a unique ID (with respect to P) and an additional attribute A. Likewise the records in Q have a unique ID (with respect to Q) and an additional attribute B.
2) A record in P whose RA and DEC coordinates are very close[4] to the RA and DEC coordinates of a record in Q is deemed to refer to the same celestial object. Thus, we can consider the *virtual table* (*virtual catalog*) of matched records between the catalogs – see Table II. The unique IDs are dropped since they play no role in the virtual catalog.

Central to the DDM algorithms we develop is the assumption that the virtual catalog can be treated as vertically partitioned (see Section II for the definition). This means that the i^{th} records in each real catalog match and form the i^{th} record in the virtual catalog. To realize

[4]in this example, identical

MATCH ID
P_1
P_2

ID	RA	DEC	A
P_1	ra_1	dec_1	a_1
P_2	ra_2	dec_2	a_2
P_3	ra_3	dec_3	a_3

MATCH ID
Q_3
Q_2

ID	RA	DEC	B
Q_1	ra_4	dec_4	b_1
Q_2	ra_2	dec_2	b_2
Q_3	ra_1	dec_1	b_4

TABLE III

CATALOGS P AND Q WITH MATCHING INDICES.

this assumption, each pair of catalogs must have co-located a distinct pair of match indices. Each index is a list of pointers; both indices have the same number of entries. The i^{th} entry in each catalogs' index points to the actual matching tuples. For example, the matching indices for catalogs P and Q above are depicted in Table III. Observe that records P_1 and Q_3 match and form the first record in the virtual catalog. Thus, P has as its first matching index, P_1, and Q has as its first Q_3. Likewise records P_2 and Q_2 match and form the second record in the virtual catalog. Thus, P has as its second matching index, P_2, and Q has as its second Q_2. Finally, since records P_3 and Q_1 do not match, they have no corresponding matching indices. Clearly, algorithms assuming a vertically partitioned virtual table can be implemented on top of these matching indices.

Creating these indices is not an easy job. Indeed, cross-matching sources is a complex problem for which no single best solution exists. The WS-CM web-service is not intended to address this problem. Instead it uses already existing solutions (*e.g.*, the cross-matching service already provided by the NVO), and is designed to allow other solutions to be plugged in easily. Moreover, cross-matching the entirety of two large surveys is a very time-consuming job and requires centralizing (at least) the RA, DEC coordinates of all tuples from both.

Importantly, the indices do not need to be created each time a data mining task is run. Instead, provided the sky survey data are static (it generally is), each pair of indices only need be created *once*. Then any data mining task can use them. In particular the DDM tasks we develop can use them. The net result is the ability to mine virtual tables at low communication cost.

IV. Data Analysis Problem: Analyzing Distributed Virtual Catalogs

We illustrate the problem with two archives: the Sloan Digital Sky Survey (SDSS) [46] and the 2-Micron All-Sky Survey (2MASS) [47].[5] Each of these has a simplified catalog containing records for a large number of astronomical point sources, upward of 100 million for SDSS and 470 million for 2MASS. Each record contains sky coordinates (RA,DEC) identifying the sources' position in the celestial sphere as well as many other attributes (460+ for SDSS; 420+ for 2MASS). While each of these catalogs individually provides valuable data for scientific exploration, together their value increases significantly. In particular, efficient analysis of the virtual catalog formed by joining these catalogs would enhance their scientific value significantly.

DEMAC addresses the data analysis problem of developing communication-efficient algorithms for analyzing user-defined subsets of virtual catalogs. The algorithms allow the user to specify a region R in the sky and a virtual catalog, then efficiently analyze the subset of tuples from that catalog with sky coordinates in R. Importantly, the algorithms we describe do not require that the base catalogs first be centralized and the virtual catalog explicitly realized. Moreover, the algorithms are not intended to be a substitute for exact, centralization-based methods currently being developed as part of the NVO [6], [7]. Rather, they are intended to complement these methods by providing, quick, communication-efficient approximate results to allow browsing. Such browsing will allow the user to better focus their exact, communication-expensive, queries.

Sections VI and VII describe some DDM algorithms that are incorporated as part of the WS-DDM web service for addressing the following problems: virtual catalog PCA and virtual catalog outlier detection. Before doing so, some PCA background is reviewed.

V. PCA Background

PCA is a well-established data analysis technique used in a large number of disciplines: astronomy, computer science, biology, chemistry, climatology, geology, *etc*. For a more detailed treatment of PCA, the reader is referred to [48].

[5]Our approach easily scales to more than two sky-surveys. However, we use this as an illustrative example for the rest of the paper.

A. PCA Theory

Given $\mathcal{X}$, an m-dimensional random vector, the goal of PCA is to find a linear mapping of $\mathcal{X}$ whose resulting m-dimensional random vector has uncorrelated components each with maximum variance. As is standard practice, we assume that $\mathcal{X}$ has mean vector zero. Let $\Sigma^{\mathcal{X}}$ denote the covariance matrix of $\mathcal{X}$, $\lambda_1^{\mathcal{X}} \geq \lambda_2^{\mathcal{X}} \geq \ldots \geq \lambda_m^{\mathcal{X}} \geq 0$ denote its eigenvalues, and $v_1^{\mathcal{X}}$, $v_2^{\mathcal{X}}$, ..., $v_m^{\mathcal{X}}$ denote their associated eigenvectors (pair-wise orthogonal and of length one). The j^{th} *principal direction* of $\mathcal{X}$ is $v_j^{\mathcal{X}}$. The j^{th} *principal component* (PC) of $\mathcal{X}$ is denoted $z_j^{\mathcal{X}}$ and equals $\mathcal{X}^T v_j^{\mathcal{X}}$. It can be shown that the PCs are pair-wise uncorrelated (have zero covariance) and capture the maximum possible variance in the following sense. For each $1 \leq j \leq m$, there does not exist $v \in \mathbb{R}^m$ orthogonal to $v_\ell^{\mathcal{X}}$ for all $1 \leq \ell < j$ such that the variance of $\mathcal{X}^T v$ is greater than the variance of $z_j^{\mathcal{X}} = \mathcal{X}^T v_j^{\mathcal{X}}$. It can further be shown that the variance of $z_j^{\mathcal{X}} = \mathcal{X}^T v_j^{\mathcal{X}}$ equals $\lambda_j^{\mathcal{X}}$.

Consider the random vector $\mathcal{Z}_{\leq r}^{\mathcal{X}} = (z_1^{\mathcal{X}}, \ldots z_r^{\mathcal{X}})^T$. If $r = m$, then $\mathcal{Z}_{\leq r}^{\mathcal{X}}$ is simply a different way of representing $\mathcal{X}$ because, $\mathcal{X} = V_{\leq r}^{\mathcal{X}} (\mathcal{Z}_{\leq r}^{\mathcal{X}})^T$ where where $V_{\leq r}^{\mathcal{X}}$ is the $n \times r$ matrix with columns $v_1^{\mathcal{X}}$, ..., $v_r^{\mathcal{X}}$. However, if $r < m$, then $\mathcal{Z}_{\leq r}^{\mathcal{X}}$ is a lossy lower dimensional representation of $\mathcal{X}$. The amount of loss is typically quantified as

$$100 \left[\frac{\sum_{j=1}^{r} \lambda_j^{\mathcal{X}}}{\sum_{j=1}^{m} \lambda_j^{\mathcal{X}}} \right], \tag{1}$$

the "percentage of variance" captured by the lower dimensional representation. The larger the percentage captured, the better $\mathcal{Z}_{\leq r}^{\mathcal{X}}$ represents the "information" contained in the original vector $\mathcal{X}$. If r is chosen so that a large percentage of the variance is captured, then, intuitively, $\mathcal{Z}_{\leq r}^{\mathcal{X}}$, captures many of the important features of $\mathcal{X}$. So, subsequent analysis on $\mathcal{Z}_{\leq r}^{\mathcal{X}}$ can be quite fruitful at revealing structure not easily found by examination of $\mathcal{X}$ directly.

B. PCA in Practise

In practice, $\Sigma^{\mathcal{X}}$ is typically not known is estimated from a dataset denoted as M, an $n \times m$ matrix with real-valued entries. The rows of M represent data records. Let M^j denote the j^{th} column and $M^j(i)$ denote the i^{th} entry of this column. Let $\mu(M^j)$ denote the *sample mean* of this column *i.e.*

$$\mu(M^j) = \frac{\sum_{i=1}^{n} M^j(i)}{n}. \tag{2}$$

To accommodate the fact that $\mathcal{X}$ is assumed to have zero mean, the dataset is standardized so that each column of M has zero sample mean. This is done by subtracting $\mu(M^j)$ from each entry in M^j. From the standardized dataset, an estimate Σ is generated for $\Sigma^{\mathcal{X}}$. Finally, the eigenvalues and their associated eigenvectors (pair-wise orthogonal and unit length) are generated from Σ: $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m \geq 0$ and v_1, v_2, ..., v_m. The j^{th} principal component is denoted z_j and equals Mv_j (the projection of M along the v_j direction). $Z_{\leq r}$, the $n \times r$ matrix with columns $z_1, \ldots, z_r$, can be thought of as a lossy lower-dimensional representation of M. The amount of loss is quantified as

$$100 \left[\frac{\sum_{j=1}^{r} \lambda_j}{\sum_{j=1}^{m} \lambda_j} \right], \tag{3}$$

the "percentage of variance" captured by the leading r PCs.

Outlier detection: Clearly the leading PCs carry valuable information. However, the lower PCs do too. Some techniques for outlier detection have been developed based on the lower PCs [48], [49], [50], [51], [52]. These techniques look to identify data records which deviate sharply from the "correlation structure" of the data. Before discussing the technical details, let us first describe what is meant by a point deviating from the correlation structure by means of an example.

Example 1: Consider a dataset where each row consists of the height and weight of a group of people (borrowed from [48] page 233). We expect that a strong positive correlation will exist – small (large) weights correspond to small (large) heights. A row with height 70 in. (175 cm) and weight 55 lbs (25 kg) may not be abnormal when height or weight is taken separately. Indeed, there may be many people in the group with height around 70 or weight around 55. However, taken together, the row is very strange because it violates the usual dependence between height and weight. □

In Example 1, the outlying record will not stand out if the projection of the data on each variable is viewed. Only when the variables are viewed together, does the outlier appear. The same idea generalizes to higher dimensional data. An outlier only stands out when all of the variables are taken into account and does not stand out over a projection onto any proper subset. Automated tests for detecting "correlation outliers" are quite valuable. The "low variance" property of the last principal components makes them useful for this job.

Recall that the j^{th} PC, $z_j = Mv_j$, has sample variance λ_j *i.e.* the variance over the entries

of z_j is λ_j. Thus, if λ_j is very small and there were no outlier data records, one would expect the entries of z_j to be nearly constant. In this case, v_j expresses a nearly linear relationship between the columns of M. A data record which deviates sharply from the correlation structure of the data will likely have its z_j entry deviate sharply from the rest of the entries (assuming no other outliers). Since the last components have the smallest $\lambda's$, then an outlier's entries in these components will likely stand out. This motivates examination of the following statistic for the i^{th} data record (the i^{th} row in M):

$$d_{1,i}^2 = z_m(i)^2 \tag{4}$$

where $z_m(i)$ denotes the i^{th} entry of z_m. A possible criticism of this approach is pointed out in [48] page 237: "it [$d_{1,i}^2$] still gives insufficient weight to the last few PCs, [...] Because the PCs have decreasing variance with increasing index, the values of $z_j(i)^2$ will typically become smaller as j increases, and $d_{1,i}^2$ therefore implicitly gives the PCs decreasing weights as j increases. This effect can be severe if some of the PCs have very small variances, and this is unsatisfactory as it is precisely the low-variance PCs which may be most effective ..."

To address this criticism, the components are normalized to give equal weight. Let w_j denote the normalized j^{th} principal direction: the $m \times 1$ vector, $\hat{v_j}$, whose i^{th} entry is $\frac{v_j(i)}{\sqrt{\lambda_j}}$. The normalized j^{th} principal component is $\hat{z_j} = Mw_j$. The sample variance of $\hat{z_j}$ equals one, so, the weights of the normalized components are equal. The statistic we use for the i^{th} data point is (following notation in [48])

$$d_{2,i}^2 = \hat{z_m}(i)^2. \tag{5}$$

Data tuples with large $d_{2,i}^2$ values can be regarded as outliers.

VI. Virtual Catalog PCA

In this section we describe two distributed algorithms for approximating the PCs on a virtual catalog, M. Both assume that the participating sites have the appropriate match indices. Hence, we describe the algorithms under the assumption that the data in each site is perfectly aligned – the i^{th} tuple of each site match (sites have exactly the same number of tuples). For simplicity, we

assume M has been vertically distributed over two sites S_A and S_B.[6] S_A has the first p attributes and S_B has the last q attributes ($p + q = m$). Let A denote the $n \times p$ matrix representing the dataset held by S_A, and B denote the $n \times q$ matrix representing the dataset held by S_B. Let $[A : B]$ denote the concatenation of the datasets *i.e.* $M = [A : B]$. Both perform the same basic two steps.

1) A distributed computation is performed to produce a sample covariance matrix, Σ (an estimate of the true covariance matrix $\Sigma^{\mathcal{X}}$). Upon completion, Σ is contained at each site.
2) Each site independently computes the eigenvalues and eigenvectors of Σ.

The algorithms differ only differ in their computation of Σ (first step).

A. Covariance Estimation Through Random Projection

This technique uses the straightforward standard formula for estimating covariance: $\Sigma = Cov(M)$ where the $(j,k)^{th}$ entry of $Cov(M)$ is

$$\frac{\sum_{i=1}^{n}[\mu(M^j) - M^j(i)][\mu(M^k) - M^k(i)]}{n-1}. \tag{6}$$

If M has been standardized ($\mu(M^j)$ subtracted from each entry in M^j), then

$$\Sigma = \frac{[A : B]^T[A : B]}{n-1}.$$

The only part of computing Σ (including standardization) requiring communication is A^TB. Hence, it suffices to estimate A^TB in a communication-efficient manner. The key idea in doing so is based on the following fact echoing the observation made in [53] (and elsewhere) that high-dimensional random vectors are nearly orthogonal.

Fact 1: Let R be an $\ell \times n$ matrix each of whose entries is drawn independently from a distribution with variance one and mean zero. It follows that $E[R^TR] = \ell I_n$ where I_n is the $n \times n$ identity matrix.

Algorithm VI-A.1 is used for computing A^TB. The result is obtained at both sites. In the communication cost calculations, we assume a number (floating point or integer) requires 4 bytes of storage.

[6]As stated earlier, this assumption can be easily relaxed.

Algorithm VI-A.1 Random Projection Based Distributed Covariance Matrix Estimation

1. S_A sends S_B a random number generator seed. **[4 bytes]**
2. S_A and S_B generate an $\ell \times n$ random matrix R. Each entry is generated independently and identically from a distribution with mean zero and variance one.
3. S_A sends RA to S_B; S_B sends RB to S_A. **[**$4m\ell$ **bytes]**
4. S_A and A_B compute $D = \frac{(RA)^T(RB)}{\ell}$.

Note that,

$$E[D] = E\left[\frac{A^T(R^TR)B}{\ell}\right] = \frac{A^TE[R^TR]B}{\ell} = A^TB. \tag{7}$$

The last equality is due to Fact 1. Hence, on expectation, the algorithm is correct. Its communication cost (bytes) divided by the cost of the centralization technique,

$$\frac{4m\ell + 4}{4nm} = \frac{\ell}{n} + \frac{1}{nm}, \tag{8}$$

is small if $\ell << n$. Indeed ℓ provides a "knob" for tuning the trade-off between communication-efficiency and accuracy. Later, in our case study, we present experiments measuring this trade-off.

B. Covariance Estimation Through Uniform Sampling

This technique simply computes $Cov(.)$ over a standardized, uniform sample of rows from M. To accomplish this, Algorithm VI-B.1 is used.

Its communication cost (bytes) divided by the cost of the centralization technique is

$$\frac{4pnm + 4}{4nm} = p + \frac{1}{nm}. \tag{9}$$

As before, p provides a "knob" for tuning the trade-off between communication-efficiency and accuracy. Later, in our case study, we present experiments measuring this trade-off.

VII. Virtual Catalog Outlier Detection

Broadly speaking, outlier detection has two important roles: data cleaning and data mining. Firstly, data cleaning is used during data preparation stages. For example, if a cross-matching

Algorithm VI-B.1 Uniform Sampling Based Distributed Covariance Matrix Estimation

1. S_A sends S_B a random number generator seed. **[4 bytes]**
2. S_A and S_B uniformly select a fraction $p \in (0, 1]$ of their tuples denoted A' and B', respectively. Note, the i^{th} tuple in A' matches the i^{th} in B', thus, $[A' : B']$ is a uniform sample of $[A : B]$.
3. S_A standardizes A' by subtracting $\mu(A^j)$ from each entry in A^j ($1 \leq j \leq p$). S_B standardizes B' by subtracting $\mu(B^j)$ from each entry in B^j ($1 \leq j \leq q$).
4. S_A sends A' to S_B; S_B sends B' to S_A. **[**$4pmn$ **bytes]**
5. S_A and S_B compute $Cov([A' : B'])$.

technique wrongly joined a pair of tuples, then the matched tuple will likely have properties making it stand out from the rest of the data. An outlier detection process might be able to identify the matched tuple and allow human operators to check for an improper match. Secondly, outlier detection can be used to identify legitimately matched tuples with unique properties. These may represent exceptional and interesting astronomical objects. Again, a human operator is alerted to further examine the situation.

In this section, we develop a distributed algorithm for outlier detection based on the PCA metric described earlier for scoring data records. Specifically, the outlier detection problem addressed is as follows. Given k (a user-specified parameter) and M a dataset vertically distributed over sites S_A and S_B, find the top k data records in M with respect to $d^2_{2,\cdot}$ (ties broken arbitrarily).[7] The algorithm we develop employs two basic steps.

1) An algorithm from Section VI is applied to approximate the last normalized principal direction, $\hat{v_m}$. At the end, each site has this approximation, denoted as $\tilde{v_m}$.
2) Let $\tilde{z_m}$ denote $M\tilde{v_m}$ and $\tilde{d}^2_{2,i}$ denote $\tilde{z_m}(i)^2$. A distributed computation is carried out to compute the top k data records with respect to $\tilde{d}^2_{2,\cdot}$.

The chief difficulty to overcome in developing a distributed algorithm for the second step is the fact that $\tilde{d}^2_{2,i}$ cannot be directly computed. Even though both sites have $\tilde{v_m}$ from the first step, $\tilde{d}^2_{2,i}$ requires information from both sites to compute. To make this point clear, let $\tilde{v_m}(A)$ and $\tilde{v_m}(B)$ denote the $p \times 1$ and $q \times 1$ vectors consisting of the first p entries of $\tilde{v_m}$ and last q

[7]As mentioned earlier, this two site assumption can be easily relaxed, but is employed for simplicity of exposition.

entries of $\tilde{v_m}$, respectively. These are the parts of $\tilde{v_m}$ corresponding to the attributes at S_A and S_B, respectively. Since $\tilde{z_m} = M\tilde{v_m} = [A : B]\tilde{v_m}$, we have

$$\tilde{z_m} = A\tilde{v_m}(A) + B\tilde{v_m}(B). \tag{10}$$

Let $\tilde{z_m^A}$ denote the first vector $(n \times 1)$ in the above sum. It can be computed entirely at S_A. Let $\tilde{z_m^B}$ denote the second vector $(n \times 1)$ in the above sum. It can be computed entirely at S_B. Letting $\tilde{z_m^A}(i)$ and $\tilde{z_m^B}$ denote the i^{th} entries, we have

$$\tilde{d}_{2,i}^2 = (\tilde{z_m^A}(i) + \tilde{z_m^B}(i))^2. \tag{11}$$

Hence, the second step above requires a distributed algorithm for computing the top k among $(\tilde{z_m^A}(1) + \tilde{z_m^B}(1))^2$, ..., $(\tilde{z_m^A}(n) + \tilde{z_m^B}(n))^2$. It suffices to solve the following problem.

Problem 1 (Distributed Sum-Square Top k (DSSTK)): Site S_A has $a_1, \ldots, a_n \in \mathbb{R}$ and S_B has $b_1, \ldots, b_n \in \mathbb{R}$. Both sites have integer $n > k > 0$. The sites must compute the top k among $c_1 = (a_1 + b_1)^2$, ..., $c_n = (a_n + b_n)^2$.

A communication-efficient algorithm for solving this problem is developed next. Unlike the algorithm for distributed PCA, this one is not approximate – its output is the correct top k. The algorithm assumes that ties are broken by choosing the larger index.[8]

A. An Algorithm for the DSSTK Problem

Developing a communication-efficient distributed algorithm which directly solves the DSSTK problem is challenging. Instead we show how solving DSSTK can be reduced to solving a simpler problem: compute the top k among $(a_1 + b_1)$, ..., $(a_n + b_n)$ (the *Distributed Sum Top k (DSTK)* Problem). Then we develop a communication-efficient algorithm for DSTK.

Let i_p^c denote the index of the p^{th} largest value among $(a_1+b_1)^2$, ..., $(a_n+b_n)^2$. Let i_p^+ denote the index of the p^{th} largest value among (a_1+b_1), ..., (a_n+b_n). Let i_p^- denote the index of the p^{th} largest value among $-(a_1+b_1)$, ..., $-(a_n+b_n)$. We have,

$$i_1^c = \begin{cases} i_1^+ & \text{if } (a_{i_1^+} + b_{i_1^+})^2 \geq (a_{i_1^-} + b_{i_1^-})^2; \\ i_1^- & \text{otherwise.} \end{cases} \tag{12}$$

[8]The choice of tie-breaking mechanism is immaterial for the algorithm. Any linear ordering will do.

Indeed, by definition, $(a_{i_1^c} + b_{i_1^c})^2 \geq (a_j + b_j)^2$ for any $1 \leq j \leq n$. Thus, if $(a_{i_1^c} + b_{i_1^c}) \geq 0$, then $(a_{i_1^c} + b_{i_1^c}) \geq (a_j + b_j)$ for any $1 \leq j \leq n$. Otherwise, $-(a_{i_1^c} + b_{i_1^c}) \geq -(a_j + b_j)$ for any $1 \leq j \leq n$. Hence, i_1^c equals i_1^+ or i_1^-. Equation (12) follows. Furthermore, by throwing away the $(i_1^c)^{th}$ values from each of the three lists and applying Equation (12), the analogous result holds for i_2^c. Repeating this process yields the following key result:

$$\text{for any } 1 \leq p \leq n,\ i_p^c = \begin{cases} i_p^+ & \text{if } (a_{i_p^+} + b_{i_p^+})^2 \geq (a_{i_p^-} + b_{i_p^-})^2; \\ i_p^- & \text{otherwise.} \end{cases} \tag{13}$$

Equation (13) implies that the DSSTK Problem is solved by carrying out the following two basic steps.

1) A distributed computation is employed, at the end of which, each site has $\langle (a_{i_p^+} + b_{i_p^+}), i_p^+ \rangle : 1 \leq p \leq k$ and $\langle (a_{i_p^-} + b_{i_p^-}), i_p^- \rangle : 1 \leq p \leq k$.
2) Each site, for $1 \leq p \leq k$, computes i_p^c according to Equation (13) and outputs $\langle (a_{i_p^c} + b_{i_p^c})^2, i_p^c \rangle : 1 \leq p \leq k$.

The second step requires no communication, so all that remains is to develop a distributed algorithm for the DSTK problem.[9]

The DSTK algorithm proceeds in rounds and its basic workings are fairly simple. (i) At the beginning of each round, each site computes its top k data values and sends their associated indices to the other site. (ii) Each site checks whether the indices it received are the exactly same (respecting ordering) as the ones it sent. If so, both sites terminate. Otherwise, (iii) both sites exchange data values for the indices received. Finally, (iv) both sites, for each data value sent or received in the previous step (say with index ℓ), update their ℓ^{th} data value to $\frac{a_\ell + b_\ell}{2}$. The sites proceed on to the next round.

The above scheme will work, but may send redundant messages. Once a data value has been updated in step (iv), its value will be identical at both sites and will never change again (even if another update is applied). Hence, these values need not be sent in future invocations of step (iii). To achieve this, both sites keep a copy of $\mathcal{U}$, a list of indices of data values that have been updated. Indices that appear in $\mathcal{U}$ need not have their data values sent during step (iii).

[9]This algorithm can then be applied again with S_A having $-a_1, \ldots, -a_n$ and S_B having $-b_1, \ldots, -b_n$ to compute $\langle (a_{i_p^-} + b_{i_p^-}), i_p^- \rangle : 1 \leq p \leq k$.

(However, the same argument does not carry over to sending indices in (i), there it is essential that repeated index transmission be allowed.)

The DSTK algorithm pseudo-code can be seen in Algorithm VII-A.1. Its not difficult to show that, due to symmetry, the copies of $\mathcal{U}$ held at each site will always be identical.

Algorithm VII-A.1 DSTK Algorithm

0. S_A sets its copy of $\mathcal{U}$ to $\emptyset$ and S_B sets its copy of $\mathcal{U}$ to $\emptyset$.

1. S_A selects the k largest among a_1, ..., a_n. Let their indices be denoted i_j^A, $1 \leq j \leq k$. S_A sends $\langle i_j^A : 1 \leq j \leq k \rangle$. In parallel, S_B selects the k largest among b_1, ..., b_n and sends $\langle i_j^B : 1 \leq j \leq k \rangle$. **[**$8k$ **bytes]**

2. Both sites check whether $i_j^A = i_j^B$ for all $1 \leq j \leq k$. If so,

   2a. for each $i_j^A \notin \mathcal{U}$, the sites exchange their $(i_j^A)^{th}$ data value **[**$\leq 8k$ **bytes]**

   2b. each site outputs $\langle (a_{i_1^A} + b_{i_1^A}), i_1^A \rangle$, ..., $\langle (a_{i_k^A} + b_{i_k^A}), i_k^A \rangle$ and terminates.

3. Otherwise, the following actions are taken. S_A sends $\langle a_{i_j^A} : 1 \leq j \leq k, i_j^A \notin \mathcal{U} \rangle$ and $\langle a_{i_j^B} : 1 \leq j \leq k, i_j^B \notin \mathcal{U} \rangle$. In parallel, S_B sends $\langle b_{i_j^B} : 1 \leq j \leq k, i_j^B \notin \mathcal{U} \rangle$ and $\langle b_{i_j^A} : 1 \leq j \leq k, i_j^A \notin \mathcal{U} \rangle$. **[**$\leq 16k$ **bytes]**

4. Both sites, for each $i_j^A, i_j^B \notin \mathcal{U}$: update their $(i_j^A)^{th}$ and $(i_j^B)^{th}$ data values to $\frac{a_{i_j^A} + b_{i_j^A}}{2}$ and $\frac{a_{i_j^B} + b_{i_j^B}}{2}$, add ℓ to their copy of $\mathcal{U}$, and goto step 1.

We illustrate the DSTK algorithm with the following example:

Example 2: Assume S_A and S_B have data values as depicted in Table IV. Let $k = 2$ *i.e.* we are interested in finding the indices of the top two among the third row.

- (Round 1:) S_A sends indices $\langle 101, 100 \rangle$; S_B sends indices $\langle 1, 2 \rangle$. Since these indices do not match, S_A replies with data values $\langle 200, 199 \rangle$ and $\langle 100, 101 \rangle$; S_B replies with data values $\langle 300, 298 \rangle$ and $\langle 203, 202 \rangle$ (note that, due to ordering, the sites can match up the data values received with their correct index). Both sites update their 1^{st}, 2^{nd}, 100^{th}, and 101^{st} data values resulting in Table V. Both sites add $1, 2, 100, 101$ to their copy of $\mathcal{U}$.
- (Round 2:) S_A sends indices $\langle 101, 100 \rangle$; S_B sends indices $\langle 101, 100 \rangle$. Since there are no mismatches (*i.e.* the first sent each site match as do the seconds), then both sites terminate each outputting data value/index pairs $\langle 403, 101 \rangle$ and $\langle 401, 100 \rangle$ as the final (correct) answer.

Note that the total communication cost divided by the cost of the centralization technique is $\frac{64}{408}$.

	1	2	3	$\cdots$	99	100	101
S_A	100	101	102	$\cdots$	198	199	200
S_B	300	298	296	$\cdots$	104	202	203
$S_A + S_B$	400	399	398	$\cdots$	302	401	403

TABLE IV

ORIGINAL DATA VALUES

	1	2	3	$\cdots$	99	100	101
S_A	200	199.5	102	$\cdots$	198	200.5	201.5
S_B	200	199.5	296	$\cdots$	104	200.5	201.5
$S_A + S_B$	400	399	398	$\cdots$	302	401	403

TABLE V

DATA VALUES AT THE END OF ROUND 1.

A substantial communications savings has been gained.

□

Correctness Proof: Now we prove that the DSTK algorithm is correct. First we show that it always terminates. Suppose the algorithm is at the start of the $(n+1)^{st}$ round. During step 1 of each of the previous rounds, at least one index selected S_A or S_B will not be in $\mathcal{U}$; otherwise, S_A and $_B$ would have selected exactly the same indices and terminated.[10] Therefore at least one index was added to $\mathcal{U}$ in each of the previous n rounds. Since indices are never removed from $\mathcal{U}$ once added, then at the start of the $(n+1)^{st}$ round, $\mathcal{U}$ will contain the indices for all data values. Thus, during step 1, S_A and $_B$ will select exactly the same indices and terminate.

Now we show that the indices outputted upon termination (i_j^*, $1 \leq j \leq k$) are correct, *i.e.* i^*j is the index of the j^{th} largest among $(a_1 + b_1)$, ..., $(a_n + b_n)$. Let $\hat{a}_1$, ..., $\hat{a}_n$ and $\hat{b}_1$, ..., $\hat{b}_n$ denote the data values held by S_A and S_B at the start of the termination round. These may not be the same as the original data values because some may have been updated during a previous round. However, it can easily be shown that for each $1 \leq \ell \leq n$, one of the following two statements holds: (i) $\hat{a}_\ell$ equals a_ℓ and $\hat{b}_\ell$ equals b_ℓ; or (ii) $\hat{a}_\ell$ and $\hat{b}_\ell$ equal $\frac{a_\ell + b_\ell}{2}$. Thus $(\hat{a}_\ell + \hat{b}_\ell)$ $= (a_\ell + b_\ell)$. Therefore, it suffices to show that $(\hat{a}_{i_j^*} + \hat{b}_{i_j^*})$ is the j^{th} largest among $(\hat{a}_1 + \hat{b}_1)$, ..., $(\hat{a}_n + \hat{b}_n)$. But this is obvious since, by definition, $\hat{a}_{i_j^*}$ and $\hat{b}_{i_j^*}$ are the j^{th} largest among $\hat{a}_1$, ...,

[10]Because the data values with indices in $\mathcal{U}$ are the same across sites, *i.e.* for any $\ell \in \mathcal{U}$, the ℓ^{th} data value for both S_A and S_B is $\frac{a_\ell + b_\ell}{2}$.

$\hat{a}_n$ and $\hat{b}_1, \ldots, \hat{b}_n$, respectively.

Communication Complexity: Since our algorithm for the DSSTK problem requires two applications of the DSTK algorithm, its total number of bytes communicated is at most $16k + 48k\rho$, where ρ is the larger number of iterations carried out between the two applications. Dividing this by the cost of the centralization technique, we get

$$\frac{16k + 48k\rho}{4n} = \frac{4k}{n} + \rho\frac{12k}{n} \tag{14}$$

Thus if $\rho << \frac{n}{12k} - \frac{1}{3}$, then a large communication savings is gained.

Synchronization Cost: Each iteration of the DSTK algorithm requires two synchronizations between sites. The two applications of DSTK can be combined such that an iteration from each is carried out using the same two synchronizations (communication cost is unaffected). Hence the total number of synchronizations carried out in solving DSSTK is ρ.

In some circumstances, reducing ρ is worthwhile, even at the expense of greater overall communication. A simple strategy for achieving this is to have each site in step 1 of DSTK send its top $\lceil \alpha k \rceil$ indices ($\alpha > 1$). Then, in steps 3 and 4, the sites will exchange their data values and update all $\lceil \alpha k \rceil$ indices. However, step 2, remains unchanged, only the top k indices are compared (otherwise the algorithm will unnecessarily solve the top $\lceil \alpha k \rceil$ problem). The communication cost per round (ignoring the termination round) increases by a factor of $\frac{24\lceil \alpha k \rceil}{24k} \approx \alpha$. But, the extra information exchanged should reduce the total number of rounds executed. We would expect, in many cases, the reduction factor to be proportional to $\frac{\lceil \alpha k \rceil}{k} \approx \alpha$.

VIII. Case Studies: Finding Galactic Fundamental Planes and Outliers

The identification of correlations among astrophysical properties has lead to important discoveries in astronomy. For example, the class of elliptical and spiral galaxies have been found to occupy a two dimensional space inside a three dimensional space of observed parameters (radius, mean surface brightness and velocity dispersion) called the *Fundamental Plane* ([54], [55]). This motivates research for finding similar correlations among other attributes in heterogeneously distributed sky surveys. Also, examination of a subset of the parameter space can also help astronomers identify objects with atypical behavior ([2],[3]). It is therefore important to systematically explore the observable parameter space, and specifically search for rare, unusual, or previously unknown types of astronomical objects and phenomena.

We perform two case studies using a virtual catalogs formed by cross-matching data from the Sloan Digital Sky Survey (SDSS) [46] and the Two Micron All Sky Survey (2MASS)[47]. This dataset is described in Section VIII-A. The first case study evaluates our distributed PCA algorithms for identifying fundamental galactic planes. It is described in Section VIII-B. The second case study evaluates our distributed outlier detection algorithm for finding outlying galaxies with respect to the "correlation structure" among the attributes in the virtual catalog. It is described in Section VIII-C.

A. Dataset

We perform experiments on a 13022 record real dataset: the *Fundamental Plane Dataset*. It serves as a virtual catalog on which we employ our distributed algorithms. However, for the purposes of our study, a real distributed environment is not necessary. Thus, for simplicity, we used a single machine and a simulated distributed environment.

Fundamental Plane Dataset: Using the web interfaces of 2MASS[11] and SDSS[12], and the SDSS object cross id tool, we obtained an aggregate dataset involving attributes from 2MASS and SDSS lying in the sky region between right ascension (RA) 150 and 200, declination (DEC) 0 and 15. The aggregated dataset had the following attributes from SDSS: Petrosian I band angular effective radius (*Iaer*), redshift (*rs*), and velocity dispersion (*vd*);[13] and had the following attribute from 2MASS: K band mean surface brightness (*Kmsb*).[14] The dataset had a 13022 tuples with four attributes. We produced a new attribute, logarithm Petrosian I band effective radius (*log(Ier)*), as log(Iaer*rs) and a new attribute, logarithm velocity dispersion (*log(vd)*), by applying the logarithm to vd. We dropped all attributes except those to obtain the three attribute dataset, log(Ier), log(vd), Kmsb.

[11]http://irsa.ipac.caltech.edu/applications/Gator/

[12]http://cas.sdss.org/astro/en/tools/crossid/upload.asp

[13]$petroRad\_i$ (galaxy view), z (SpecObj view) and velDisp (SpecObj view) in SDSS DR4

[14]$k\_mnsurfb\_eff$ in the extended source catalog in the All Sky Data Release, http://www.ipac.caltech.edu/2mass/releases/allsky/index.html

B. Case Study: Finding Galactic Fundamental Planes

This case study aims to identify to what extent the galaxies represented in the two datasets can be thought to reside on a fundamental plane. This amounts to determining the percentage of variance captured by the first two principal components. Both of distributed PCA algorithms have a parameter allowing us to specify exactly how much communication they require. For each value of this parameter on each dataset, we employ each algorithm 100 times computing the percentage of variance captured by the first two PCs. We report averages and 0.95 confidence intervals (error bars).

We compare the distributed algorithms against the centralization technique which simply sends all the data to a single location and performs PCA there. In our experiments, we measure the accuracy of the distributed algorithms as a function of the amount of communication they require. Communication required is measured as a percentage:

$$\text{Communication Percentage } = 100\frac{\text{Bytes required by distributed algo}}{\text{Bytes required by centralization technique}}.$$

The communication percentage of the random projection based distributed PCA algorithm (*RandProj*) is 100 times Equation (8). The communication percentage of the uniform sampling distributed PCA algorithm (*UnifSamp*) is 100 times Equation (9).

Figure 2 shows the communication costs versus accuracy of RandProj and UnifSamp on the fundamental plane dataset. As can be seen, UnifSamp clearly outperforms RandProj. At low communication percentages, UnifSamp produces results very close to the centralized. Take note that RandProj does approach the centralized results given large enough communication. It is surprising that it initially moves away from the centralized results. We cannot explain this observation.

Based on these results, it would seem that using UnifSamp to estimate the covariance matrix for outlier detection would be better than RandProj. However, since outlier detection is based on the PCs and not the variances captured, it is not entirely clear which covariance matrix estimation is preferred. In the next subsection we directly examine this issue.

C. Case Study: Finding PCA Based Outlying Galaxies

This case study aims to identify the top k outlying galaxies with respect to the PCA-based outlier score, $d^2_{2,\cdot}$, described earlier. The centralized technique first computes the third principal

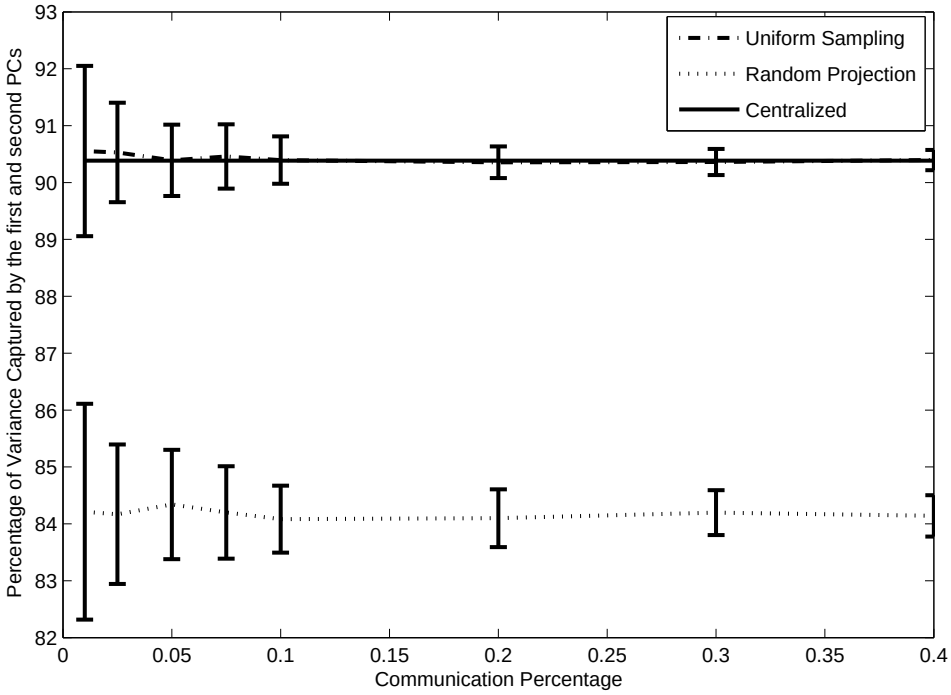


Fig. 2. Communication percentage vs. the percentage of variance captured by the first two PCs on the fundamental plane dataset. "Centralized" refers to the percentage of variance captured by the first two PCs using the centralized technique.

direction (both our datasets have three attributes). Then, computes $d^2_{2,\cdot}$ for each data record and returns the top k along with their score. On each dataset, we compare the outliers reported with those reported by our distributed outlier detection technique using UnifSamp in the PCA phase. As before, for each parameter setting, we repeat the outlier detection algorithm 100 times. There are two questions to address before describing our results: How do we measure communication percentage? How do we measure accuracy?

Measuring communication percentage and accuracy: The communication cost of the distributed outlier detection algorithm is the cost of the uniform sampling PCA phase plus the cost of the DSSTK phase. Thus, the communication percentage is (assuming a virtual catalog with n records and m attributes)

$$100\left[\frac{(4pnm+4)+(16k+48k\rho)}{4mn}\right] = 100\left[p+\frac{1}{nm}+\frac{4k}{mn}+\frac{12k\rho}{mn}\right]. \tag{15}$$

Accuracy is more difficult to measure. Let $s_1, \ldots s_n$ be true outlier scores of the records, *i.e.* the scores computed by the centralized technique, written in descending order. Let $\hat{s}_1, \ldots, \hat{s}_n$ denote their corresponding scores if estimated using the uniform sampling based distributed PCA algorithm. Let $\hat{i}_p$ be the index of the p^{th} largest among $\hat{s}_1, \ldots, \hat{s}_n$. Ultimately, we would like to quantify the extent to which the records, $r_{\hat{i}_1}, \ldots, r_{\hat{i}_k}$, form a good approximation to the true top k records, $r_1, \ldots, r_k$.

Fagan *et al.* [56] discuss the problem of defining measures for comparing two top k lists. One of the measures they consider is a generalization of Kendall's tau from statistics. Adopting notation similar to theirs, let D denote the set of true top k records $\{r_1, \ldots, r_k\}$; let $\hat{D}$ denote the set of estimated top k records $\{r_{\hat{i}_1}, \ldots, r_{\hat{i}_k}\}$; let z denote $|D \cap \hat{D}|$; and let $\mathcal{P}$ denote the set of all size two subsets of $(D \cup \hat{D})$. For each pair $\{r_i, r_j\}$ in $\mathcal{P}$, let $\overline{K}(i,j)$ denote the "penalty" assigned to this pair. If the true and estimated top k lists are inconsistent with respect to this pair, then the penalty is set to one, otherwise zero. Three situations are deemed to cause an inconsistency.

1) r_i and r_j appear in both top k lists. Moreover, the records appear in different orders in the two lists. r_i comes before r_j in true top k list but comes after in the estimated top k list; or, r_i comes after r_j in true top k list but comes before in the estimated top k list.
2) r_i and r_j both appear in one of the top k lists and exactly one of the records (say r_i) appears in the other list. Moreover, r_i comes after r_j in the list in which they both appear.
3) r_i and r_j each appear in exactly one of the lists and both do not appear in the same list.

The final, unnormalized measure is defined to be the sum of the penalties[15]

$$\sum_{\{r_i, r_j\} \in \mathcal{P}} \overline{K}(i,j).$$

Following from Lemma 3.1 in [56], this sum is tightly bounded from above by $(k-z)(k+z) + \frac{z(z-1)}{2}$. Hence, the final, normalized measure is[16]

[15]Denoted $K^{(0)}(\tau_1, \tau_2)$ in [56].

[16]In our prior work [45] we used a similar, but more ad-hoc, measure because we were unaware of [56]. However, we drop that measure here in favor of Krus.

$$Krus = \frac{\sum_{\{r_i,r_j\}\in\mathcal{P}} \overline{K}(i,j)}{(k-z)(k+z)+\frac{z(z-1)}{2}}. \tag{16}$$

We use this as one measure of accuracy in our experiments. However, we feel the measure has weaknesses related to the fact that it does not take into account the actual outlier scores, only the top k records themselves. Hence there are situations where Krus gives value very close to one (indicating poor estimation), but a plausible argument can be made that the estimation is actually decent.

Example 3: Suppose $s_1 - s_{2k} \leq \epsilon$ and the estimated top k records are r_{k+1}, ..., r_{2k}, *i.e.* $\hat{i}_1 = k+1$, ..., $\hat{i}_k = 2k$. In this case, Krus returns one *regardless* of the value of ϵ. However, when ϵ is very close to zero, it seems intuitive that any k records out of r_1, ..., r_{2k} are a decent top k since they all have nearly the same outlier score. □

We would like to use an accuracy measure that takes into account the actual outlier scores. However, to our knowledge, no such measure for comparing top k lists has been developed in the literature. Therefore, we develop one of our own. Since we are assuming a true top k, we consider only records r_j ($1 \leq j \leq k$) and develop a penalty for its estimate $r_{\hat{i}_j}$. We choose not to assign a penalty to all pairs as Fagan *et al.* do, because they were interested in comparing to top k lists without assuming one was the true answer.

We also do not consider the order in which the true top k appear, *i.e.*, if $r_1, \ldots r_k$ and $r_{\hat{i}_1}, \ldots, r_{\hat{i}_k}$ differ only in the order in which the records appear, then we assume no error has occurred.[17] With this in mind, the penalty for estimating r_j as $r_{\hat{i}_j}$, denoted R_j, is zero if $r_{\hat{i}_j}$ is among the true top k, *i.e.*, $s_{\hat{i}_j} \geq s_k$. Otherwise, the penalty is amount by which the outlier score of $r_{\hat{i}_j}$ would need be increased to be in the true top k, *i.e.*, $s_k - s_{\hat{i}_j}$. Formally stated, the penalty is defined as

$$R_j = \begin{cases} 0 & \text{if } s_{\hat{i}_j} \geq s_k; \\ s_k - s_{\hat{i}_j} & \text{otherwise.} \end{cases} \tag{17}$$

An overall, unnormalized accuracy metric, is

$$\sum_{j=1}^{k} R_j. \tag{18}$$

[17]We make this assumption to simplify normalization.

Observe that, for ϵ very close to zero in Example 3, (18) is very close to zero, matching our intuition. Finally, to improve interpret-ability, we normalize (18). Under the assumption that $s_k > s_{n-k+1}$ (which holds in all experiments we run), (18) is bounded tightly above by $\sum_{j=1}^{k}(s_k - s_{n-j+1})$. Hence, the normalized accuracy measure we use, and refer to as *Outlier Score Error (OSE)*, is

$$OSE = \frac{\sum_{j=1}^{k} R_j}{\sum_{j=1}^{k}(s_k - s_{n-j+1})}. \tag{19}$$

In our experiments we used both accuracy measures Krus and OSE.

Experimental results: Figure 3 shows the results using the both accuracy measures and $K = 5, 10, 50$.

UnifSamp clearly outperforms RandProj with respect to both accuracy measures and all values of K. Moreover, RandProj appears to level-off at a non-zero (in most cases, significantly large) degree of error. This observation is consistent with the results in the previous case study. Beyond this however, we cannot explain this surprising observation.

UnifSamp will result in zero error once 100 percent communication is reached since the sampling is done without replacement. The rate of convergence is much faster with respect to the OSE measure. We believe this is due to Krus presenting an overly pessimistic view by not taking into account the actual outlier scores (only the top k records themselves). As such, there are situations where Krus gives value very close to one (indicating poor estimation), but a plausible argument can be made that the estimation is actually decent (see Example 3). OSE, on the other hand, takes into account the actual outlier scores, and it shows that, at low communication percentages, UnifSamp produces results very close to the centralized.

IX. Conclusions

The challenges of scientific data analysis within the astronomy community are growing ever more difficult. This is primarily a result of rapidly growing data volumes from a growing number of specialty missions (e.g., wavelength-specific space observatories in Astronomy; or helio/geolocation-specific plasma sensing instruments for Space Physics; or domain-specific remote sensing instruments for Earth System Science). Data analysis for these multiple missions is growing in complexity and difficulty for several reasons: (a) the data are often distributed

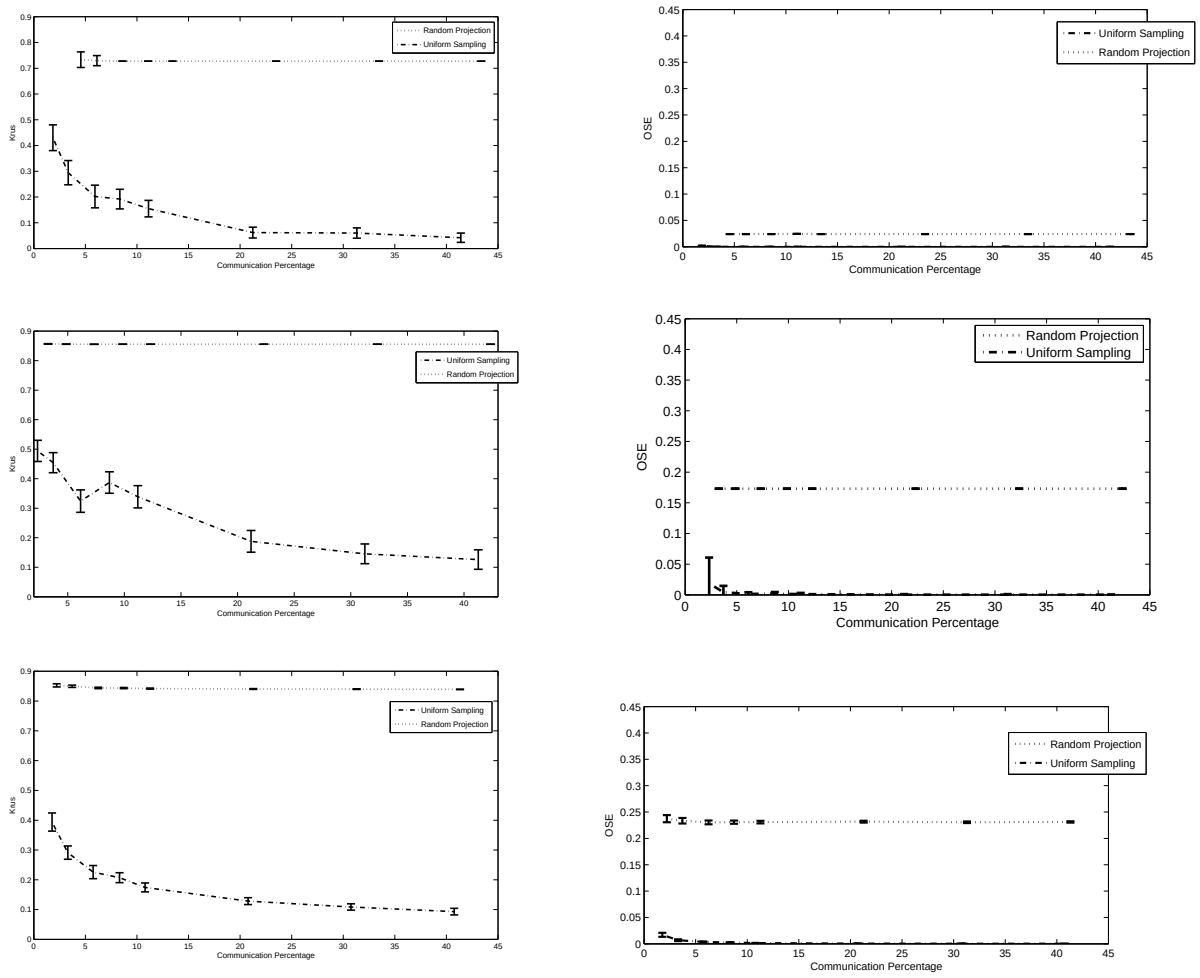


Fig. 3. Communication percentage vs. Krus accuracy: Krus measure (left column), OSE measure (right column). The rows from top to bottom depict results with $K = 5, 10, 50$.

across different science data centers (e.g., the distributed Science Archive Research Centers, SARCs, for Astronomy); (b) science instruments, their data-gathering modes, and their data products are becoming increasingly complex; and (c) the data volumes are growing rapidly. Hence, the greatest challenge to reaping the maximum scientific benefit from this wealth of data is to develop *distributed* data mining algorithms.

In this paper, we described a system for the Distributed Exploration of Massive Astronomical Catalogs (DEMAC). It is built on top of the existing U.S. National Virtual Observatory environment and provides tools for data mining (as web services) without requiring datasets to be down-loaded to a centralized server. In order to test the functionality of the system, we developed

algorithms for distributed PCA. Using the approximate PCs, we designed a communication-efficient distributed algorithm for outlier detection. We carried out two case studies for estimating principal components and outliers in the Fundamental Plane of astronomical parameters. We envision DEMAC to increase the ease with which large, geographically distributed astronomy catalogs are explored and astronomers can better tap the riches of distributed virtual sky survey catalogs.

X. Acknowledgments

This work is supported in part by U.S. National Science Foundation (NSF) awards IIS-0093353 (CAREER), IIS-0329143, and by U.S. National Aeronautics and Space Administration (NASA) award NNX07AV70G.

References

[1] "US National Virtual Observatory," http://www.us-vo.org/.

[2] Zhang X, Luo A.L. and Zhao Y.H., "Outlier Detection in Astronomical Data," *Proceedings of the SPIE*, vol. 5493, pp. 521–529, 2004.

[3] Djorgovski S. G.,Gal R. R, Mahabal A, Brunner R, Castro S. M, Odewahn S. C, de Carvalho R. R, , "DPOSS Team. Color-Space Outliers in DPOSS: Quasars and Peculiar Objects," *Bulletin of the American Astronomical Society*, vol. 32, pp. 1600–, 2000.

[4] "Digital Dig - Data Mining in Astronomy," http://www.astrosociety.org/pubs/ezine/datamining.html.

[5] "NASA's Data Mining Resources for Space Science," http://rings.gsfc.nasa.gov/∼borne/nvo_datamining.html.

[6] "Framework for Mining and Analysis of Space Science Data," http://www.itsc.uah.edu/f-mass/.

[7] "The ClassX Project: Classifying the High-Energy Universe," http://heasarc.gsfc.nasa.gov/classx/.

[8] "The AUTON Project," http://www.autonlab.org/autonweb/showProject/3/.

[9] "Scientific Data Mining, Integration and Visualization Workshop," http://www.anc.ed.ac.uk/sdmiv/.

[10] B. H. Park and H. Kargupta, "Distributed data mining: Algorithms, systems, and applications," in *Data Mining Handbook*. To be published, 2002. [Online]. Available: http://www.cs.umbc.edu/~hillol/pubs.html

[11] Kargupta H. and Sivakumar K., "Existential Pleasures of Distributed Data Mining," in *Data Mining: Next Generation Challenges and Future Directions, MIT/AAAI Press*, 2004, pp. 3–26.

[12] "Distributed Data Mining in Astrophysics and Astronomy," http://www.cs.queensu.ca/home/mcconell/DDMAstro.html.

[13] "GRIST: Grid Data Mining for Astronomy," http://grist.caltech.edu.

[14] "International Virtual Observatory," http://www.ivoa.net.

[15] "EU Data Grid," http://web.datagrid.cnr.it/.

[16] "Griphyn – Grid Physics Network," http://www.griphyn.org/.

[17] “International Virtual Data Grid Laboratory,” http://www.ivdgl.org/.

[18] “Particle Physics Data Grid,” http://www.ppdg.net/.

[19] “Astrophysical Virtual Observatory,” http://www.euro-vo.org/.

[20] “US National Virtual Observatory,” http://us-vo.org/.

[21] “NASA’s Information Power Grid,” http://www.ipg.nasa.gov/.

[22] “bioGrid – Biotechnology Information and Knowledge Grid,” http://www.bio-grid.net/.

[23] “Data Mining Grid,” http://www.datamininggrid.org/.

[24] “Knowledge Grid Lab,” http://dns2.icar.cnr.it/kgrid/.

[25] “GridMiner,” www.gridminer.org.

[26] Hinke T. and Novotny J., “Data Mining on NASA’s Information Power Grid,” in *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC’00)*, 2000, p. 292.

[27] Hofer J. and Brezany P., “Distributed Decision Tree Induction Within the Grid Data Mining Framework,” in *Technical Report TR2004-04, Institute for Software Science, University of Vienna*, 2004.

[28] “First International Workshop on Knowldge and Data Mining Grid (KDMG’05),” http://laurel.datsi.fi.upm.es/KDMG05/.

[29] “Workshop on Data Mining and the Grid (DM-Grid 2004),” http://www.cs.technion.ac.il/∼ranw/dmgrid/program.html.

[30] Park B. and Kargupta H., “Distributed Data Mining: Algorithms, Systems, and Applications,” in *The Handbook of Data Mining, Lawrence Erlbaum Associates*, 2003, pp. 341–358.

[31] Kargupta H., Huang W., Sivakumar K., and Johnson E., “Distributed clustering using collective principal component analysis,” *Knowledge and Information Systems Journal*, vol. 3, pp. 422–448, 2001.

[32] Kargupta H. and Puttagunta V., “An Efficient Randomized Algorithm for Distributed Principal Component Analysis from Heterogeneous Data,” in *Proceedings of the Workshop on High Performance Data Mining in conjunction with the Fourth SIAM International Conference on Data Mining*, 2004.

[33] Johnson E. and Kargupta H., “Collective, Hierarchical Clustering From Distributed, Heterogeneous Data,” in *Lecture Notes in Computer Science*, M. Zaki and C. Ho, Eds., vol. 1759. Springer-Verlag, 1999, pp. 221–244.

[34] Chen R. and Sivakumar K., “A New Algorithm for Learning Parameters of a Bayesian Network from Distributed Data,” *Proceedings of the Second IEEE International Conference on Data Mining (ICDM)*, pp. 585–588, 2002.

[35] Chen R., Sivakumar K., and Kargupta H., “Learning Bayesian Network Structure from Distributed Data,” in *Proceedings of the Third SIAM International Conference on Data Mining*, 2003, pp. 284–288.

[36] Caragea D., Silvescu A., and Honavar V., “Decision Tree Induction from Distributed Data Sources,” in *Proceedings of the Conference on Intelligent Systems Design and Applications*, 2003.

[37] Giannella C., Liu K., Olsen T., and Kargupta H., “Communication Efficient Construction of Decision Trees Over Heterogeneously Distributed Data,” in *Proceedings of the The Fourth IEEE International Conference on Data Mining (ICDM)*, 2004.

[38] Hershberger, D. and Kargupta, H., “Distributed Multivariate Regression Using Wavelet-based Collective Data Mining,” *Journal of Parallel and Distributed Computing*, vol. 61, pp. 372–400, 1999.

[39] Park B., Kargupta H., Johnson E., Sanseverino E., Hershberger D., and Silvestre L., “Distributed, Collaborative Data Analysis From Heterogeneous Sites Using a Scalable Evolutionary Technique,” *Applied Intelligence*, vol. 16, no. 1, 2002.

[40] Tumer K. and Ghosh J., “Robust Order Statistics Based Ensembles for Distributed Data Mining,” in *Advances in Distributed and Parallel Knowledge Discovery*. MIT/AAAI Press, 2000, pp. 185–210.

[41] Chris Giannella, Haimonti Dutta, Ran Wolff, Kirk Borne and Hillol Kargupta., “Distributed Data Mining in Astronomy

Databases," in *In the 9th Workshop on Mining Scientific and Engineering Data Sets, as part of the SIAM International Conference on Data Mining (SDM)*, 2006.

[42] Hodge V. and Austin J., "A Survey of Outlier Detection Methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, 2004.

[43] Barnett V. and Lewis T., *Outliers in Statistical Data*. John Wiley & Sons, 1994.

[44] Hung E. and Cheung D., "Parallel Mining of Outliers in Large Database," *Distributed and Parallel Databases*, vol. 12, pp. 5–26, 2002.

[45] Haimonti Dutta, Chris Giannella, Kirk Borne and Hillol Kargupta., "Distributed Top-K Outlier Detection from Astronomy Catalogs using the DEMAC System," in *Proceeding of the SIAM International Conference on Data Mining (SDM)*, 2007.

[46] "Sloan Digital Sky Survey," http://www.sdss.org.

[47] "2-Micron All Sky Survey," http://pegasus.phast.umass.edu.

[48] Jolliffe I., *Principal Component Analysis*. Springer-Verlag, 2002.

[49] Gnanadesikan R. and Kettenring J., "Robust Estimates, Residuals, and Outlier Detection with Multiresponse Data," *Biometrics*, vol. 28, pp. 81–124, 1972.

[50] Hawkins D. and Fatti P., "Exploring Multivariate Data Using the Minor Principal Components," *The Statistician*, vol. 33, pp. 325–338, 1984.

[51] Hawkins D., "The Detection of Errors in Multivariate Data Using Principal Components," *Journal of the American Statistical Association*, vol. 69, no. 346, pp. 340–344, 1974.

[52] Shyu M.-L., Chen S.-C., Sarinnapakorn K., and Chang L., "A Novel Anomaly Detection Scheme Based on a Principal Component Classifier," in *Proceedings of the Foundations and New Directions of Data Mining Workshop, in Conjuction with the Thrid IEEE International Conference on Data Mining (ICDM)*, 2003, pp. 172–179.

[53] K. Liu, H. Kargupta, and J. Ryan, "Random projection-based multiplicative data perturbation for privacy preserving distributed data mining," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 18, no. 1, pp. 92–106, 2006.

[54] "Elliptical Galaxies: Merger Simulations and the Fundamental Plane," http://irs.ub.rug.nl/ppn/244277443.

[55] J. L. and C. W., "A Statistical Comparison of Line Strength Variations in Coma and Cluster Galaxies at $z \sim 0.3$," *Astronomical Society, Australia*, vol. 15, pp. 309–317, 1998.

[56] Fagan R., Kumar R., and Sivakumar D., "Comparing Top K Lists," *SIAM Journal on Discrete Mathematics*, vol. 17, no. 1, pp. 134–160, 2003.

Distributed Linear Programming and Resource Management for Data Mining in Distributed Environments

Submitted for Blind Review

Abstract

Advances in computing and communication has resulted in very large scale distributed environments in recent years. They are capable of storing large volumes of data and often have multiple compute nodes. However, the inherent heterogeneity of data components, the dynamic nature of distributed systems, the need for information synchronization and data fusion over a network and security and access control issues makes the problem of resource management and monitoring a tremendous challenge. In particular, centralized algorithms for management of resources and data may not be sufficient to manage complex distributed systems. In this paper, we present a distributed algorithm for resource and data management which builds on the traditional simplex algorithm used for solving linear optimization problems. Our distributed algorithm is an exact one meaning its results are identical if run in a centralized setting. We provide extensive analytical results and experiments on simulated data to demonstrate the performance of our algorithm.

1. Introduction

Linear programs with constraints have been used extensively in data mining applications. For example, in Support Vector Machines (SVMs) with expert knowledge [24] a linear program is formulated such that the constraints correspond to prior knowledge. Such systems are called *knowledge based linear programs* and can often end up with an infinite number of constraints, resulting in a semi-infinite linear program [23]. However, Mangasarian *et al.* established that the these semi-infinite linear programs and linear programs with equilibrium constraints can be solved as *single linear programs* with a finite number of constraints, thus leading to a simple linear programming formulation. Although, this is a relatively new area of research, several other real world applications can be modeled as knowledge based linear or convex programs.

Consider for example, a real time command and control system such as the Ballistic Missile Defense System (BMDS) designed by the Missle Defense Agency[1]. Such a system is fundamentally distributed, near real time and should have the capability to manage, correlate and assess information obtained from advanced sensors, lasers, command and control devices and battle management and planning. Such decentralized applications are faced with the critical and challenging problem of resource and data management. This problem can be posed as a linear optimization problem [1]. In such distributed applications, it is possible to either transfer data to a central site and then do the mining or to build local distributed models. A decision must be made regarding the optimal mechanism of data transfer in the network such that the best balance between communication cost and accuracy is attained. However, finding a *centralized* site willing to solve this optimization problem may be difficult due to security and access control issues in the network. For instance, in the BMDS system, modular designs of monitoring the architecture are encouraged since security of individual components is as critical as the state of these components collectively across the BMDS. In this paper, we present a distributed strategy for solving the linear optimization problem which is very useful for resource management in DDM applications.

This paper is organized as follows: Section 2 describes related work; Section 3 reviews a popular methodology for solving linear programs namely the Simplex Algorithm; Section 4 outlines the model of the distributed data mining environment we are considering; Section 5 describes the Distributed Simplex Algorithm and Section 6 presents empirical results on simulated data. Finally, Section 7 concludes the paper and discusses future work in the area.

2 Related Work

This section presents a brief overview of resource discovery in distributed environments and some well known optimization techniques.

[1] http://www.mda.mil/mdalink/html/mdalink.html

2.1 Resource Discovery in distributed environments

In recent years, a lot of research has been done on resource discovery in distributed environments. Iamnitchi et. al [9] state that the two resource sharing environments that are of particular interest to user communities are grids and peer-to-peer (P2P) systems. They refer to the four main components of a resource discovery solution as membership protocol, overlay construction function, preprocessing and local or remote request processing. Matchmaking [10] and distributed cycle sharing [11] are two other distributed resource management mechanisms. Our work differs from all the above in that our objective is to design a distributed optimization technique for efficient management of data transfers.

2.2 Optimization Techniques

One of the most popular algorithms for linear programming is the simplex algorithm [2]. We discuss Dantzig's algorithm in detail in section 3. However, this is not the only way to solve a linear program. The main competitors are a group of methods known as interior point methods ([12] and the references therein). These algorithms have been inspired by Karmarkar's algorithm [13]. As opposed to the simplex method, interior point methods reach the optimal vertex by traversing the interior of the feasible region. Some interior point methods have polynomial worst case running times, which are less than the exponential worst case running time of the simplex method. On average, however, the simplex method is competitive with these methods.

Several other parallel implementations of the linear optimization algorithms exist [14, 15, 16, 17]. Stunkel and Reed [16] consider two different approaches of parallelization of the constraint matrices on the hypercube: (1) Column partitioned simplex and (2) Row partitioned simplex. Column partitioned simplex have been studied further in work done by Yarmish [14]. Their parallel algorithm divides the columns of the constraint matrix among many processors. Ho and Sundaraj [17] compare the problem of distributed computation of the simplex method using two different methods: (1) Distributed Reinversion (DINV) and (2) Distributing Pricing (DPRI). Eckstein et. al. [19] present parallel implementations of the simplex algorithm using computational devices called "stripe arrays" which resemble temporary data structures used in some routines of Connection Machine Scientific Software Library, CMSSL.

In the following section we present a review of the Simplex Algorithm.

3 A Review of the Simplex Algorithm

Mathematically, the linear optimization problem [2] can be stated as follows (using vector notation):

Find $x_1 \geq 0, x_2 \geq 0, \cdots, x_n \geq 0$ and the minimum value (z) of the objective function:

$$c_1x_1 + c_2x_2 + \cdots + c_nx_n \tag{1}$$

satisfying the constraints:

$$A_1x_1 + A_2x_2 + \cdots + A_nx_n = B \tag{2}$$

where $A_j = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{pmatrix}$ and $B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$.

For the maximization problem, the negative of the objective function can be minimized. The $m \times n$ linear system defines the constraints on the objective function i.e. each row of the matrix A defines a constraint on n variables. Moreover, the constraints and objective function can be represented by a *"simplex tableau"* (terminology adapted from the optimization literature). The Table 1 gives an illustration.

a_{11}	a_{12}	$\cdots$	a_{1n}	b_1
a_{21}	a_{22}	$\cdots$	a_{2n}	b_1
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
a_{m1}	a_{m2}	$\cdots$	a_{mn}	b_m
c_1	c_2	$\cdots$	c_n	z

Table 1. Simplex Tableau

A *Basic Solution* to the linear optimization problem is obtained by setting $n - m$ variables equal to zero and solving the resulting $m \times m$ linear system of equations. In a basic solution, the n - m variables at zero are called *non-basic* variables, while the remaining m variables are called *basic variables.* If all of the basic variables take non negative values, then the basic solution is called a *Basic Feasible Solution* (BFS).

A fundamental theorem in linear programming concerns the replacement of a linear system, if possible, by the equivalent **canonical** system. This allows one to identify BFS's, and to move from one BFS's to another easily through a "pivot operation". Quoting [2]: "A *canonical* system with an ordered subset of variables, called *basic* is a system such that for each i, the i^{th} basic variable has a unit coefficient in the i^{th} equation and has zero coefficients elsewhere." To identify the BFS from a canonical form, all of the non-basic variables are set to zero and values for the basic variables are simply read-off. The simplex algorithm decreases the

value of the objective function at each iteration by selecting a basic variable and replacing the corresponding of column of matrix A with another column.

In most practical linear programming problems, the equation $Ax = b$ takes the form of an inequality. In order to transform these inequalities into equations, additional variables are introduced as follows: (1) If $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$ a *slack* variable is introduced: $\sum_{j=1}^{n} a_{ij}x_j + s_i = b_i$. (2) If $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$ a *surplus* variable and an *artificial* variable are added: $\sum_{j=1}^{n} a_{ij}x_j - s_i + r_i = b_i$. (3) If $\sum_{j=1}^{n} a_{ij}x_j = b_i$ an *artificial* variable is added: $\sum_{j=1}^{n} a_{ij}x_j + r_i = b_i$. It must be noted that that artificial variables must be zero when the optimum is reached; however this is not required of slack and surplus variables.

It is required for the simplex algorithm that m of the vectors A_j be independent. Let A_{j_1}, A_{j_2}, $\cdots$, A_{j_m} be such a set of independent vectors forming a basis P i.e.

$$P = \left[A_{j_1}, A_{j_2}, \cdots, A_{j_m}\right] \tag{3}$$

A canonical system can be obtained by multiplying equation 2 by P^{-1} i.e.

$$(P^{-1}A_1)x_1 + (P^{-1}A_2)x_2 + (P^{-1}A_3)x_3 + \cdots + (P^{-1}A_n)x_n = (P^{-1}B)$$

or,

$$\hat{A_1}x_1 + \hat{A_2}x_2 + \cdots + \hat{A_n}x_n = \hat{B} \tag{4}$$

where $\hat{A_j} = P^{-1}A_j$ and $\hat{B} = P^{-1}B$. Note that $P^{-1}P = I$ and $\hat{A_{j_i}} = U_i$ where U_i is a unit vector with unity in component i and zero elsewhere. Since equation 4 is in canonical form with basic variables $x_{j_1}, x_{j_2}, \cdots, x_{j_m}$ the basic solution may be obtained by setting non-basic variables to 0. Hence,

$$\begin{pmatrix} x_{j_1} \\ x_{j_2} \\ \vdots \\ x_{j_m} \end{pmatrix} = P^{-1}B \tag{5}$$

The basic solution is feasible if $\hat{B} \geq 0$ (This implicitly means that each component of $\hat{B}$ is greater than or equal to 0. The next step is to eliminate the x_{j_i}'s from the equation 1 producing the *relative cost factors* $\hat{c_j}$. Let us define γ as in equation 6.

$$\gamma = \left[c_{j_1}, c_{j_2}, \cdots, c_{j_m}\right] \tag{6}$$

Multiplying equation 4 by γ yeilds the following:

$$(\gamma\hat{A_1})x_1 + (\gamma\hat{A_2})x_2 + \cdots + (\gamma\hat{A_n})x_n = (\gamma\hat{B}) \tag{7}$$

Note that $\gamma\hat{A_j}$ = γU_i = c_{j_i} so that equation 7 has the same coefficients for the basic variables as equation 1. Elimination of basic variables results in the following equations:

$$(c_1 - \gamma\hat{A_1})x_1 + (c_2 - \gamma\hat{A_2})x_2 + \cdots + (c_n - \gamma\hat{A_n})x_n = (z - \gamma\hat{B}) \tag{8}$$

$$\text{or, } \hat{c_1}x_1 + \hat{c_2}x_2 + \cdots + \hat{c_n}x_n = (z - \gamma\hat{B}) \tag{9}$$

Also,

$$(c_1 - \gamma P^{-1}A_1)x_1 + (c_2 - \gamma P^{-1}A_2)x_2 + \cdots + (c_n - \gamma P^{-1}A_n)x_n = (z - \gamma P^{-1}B)$$

which implies,

$$(c_1 - \pi A_1)x_1 + (c_2 - \pi A_2)x_2 + \cdots + (c_n - \pi A_n)x_n = (z - \pi B)$$

where $\pi = \gamma P^{-1}$ and $\hat{c_j} = c_j - \gamma\hat{A_j}$. It can be clearly seen that, the relative costs are obtained by subtracting from c_j a weighted sum of the coefficients $a_{1j}, a_{2j}, \cdots, a_{mj}$. Also, the weights $\pi_1, \pi_2, \cdots, \pi_m$ are the components of π and can be assumed to be equal for all j. Each of the π_j are referred to as the *simplex multipliers*.

Since,

$$\begin{aligned} \pi &= \gamma P^{-1} \\ \text{Multiplying by P gives } \pi P &= \gamma \\ \text{or, } \pi(P_{j_1}, P_{j_2}, \cdots, P_{j_m}) &= (c_{j_1}, c_{j_2}, \cdots, c_{j_m}) \\ \text{Hence, } \pi P_{j_i} &= c_{j_i} \end{aligned}$$

The basic solution is optimal if $\hat{c_j} \geq 0$. If not all $\hat{c_j} \geq 0$ then an improved solution can be obtained by choosing s such that

$$\hat{c_s} = \text{Min } \hat{c_j} \tag{10}$$

The corresponding column of the A matrix is referred to as the *pivot column*. Next a column must be chosen to leave the basis such that replacing it with the entering basis column still ensures that the criteria for basic feasible solution still holds. Recall that $\hat{B} = P^{-1}B$ and $\hat{B} = \begin{bmatrix} \hat{b_1} \\ \hat{b_2} \\ \vdots \\ \hat{b_m} \end{bmatrix}$ and $\hat{P_s} = \begin{bmatrix} \hat{a_{1s}} \\ \hat{a_{2s}} \\ \vdots \\ \hat{a_{ms}} \end{bmatrix}$ In order to guarantee that a new basic feasible solution will be reached we do the following:

$$\text{Find row i that minimizes } \frac{\hat{b_i}}{\hat{a_{is}}} \tag{11}$$

This row i is referred to as the *pivot row*.

This completes one iteration of the simplex algorithm. Next, we discuss the computational cost of the Simplex algorithm.

3.1 Computational Complexity

The performance of the simplex method is usually measured in terms of the number of pivots required to solve the linear programming problem [3]. Theoretically, an upper bound on the number of pivots was first obtained by Dantzig [2] where he showed that for an $m \times n$ Linear Programming problem, there are at most $\binom{n}{m}$ bases and hence this is the maximum number of pivots. But, $\binom{n}{m}$ is a very large number. When $n = 2m$ for example, it grows as $\frac{2^{2m}}{\sqrt{\pi m}} > (\frac{n}{m} - 1)^m$ which is exponential in m for $\frac{n}{m} > 2$. This has been further studied by Klee and Minty [5].

We stress that our objective here is not to perform a comprehensive study of the complexity[2] of the simplex algorithm – rather we use this to illustrate how a distributed optimization algorithm can be designed effectively. In the following section we describe the structure of a distributed data mining application and formulate the linear optimization problem we are interested in solving.

4 The Distributed Data Model

The DDM application can be conceptualized as a weighted graph G = (V, E) where V represents the node set and E the edge set that connect pairs of nodes. For our purposes, the graph is assumed to be undirected and has a fixed topology. We also assume that communication between nodes is completely reliable.

Formally, we assume that there are n different nodes in the network. Each node has a dataset D_i residing on it[3]. The cost of processing data at the i^{th} node into a data mining model is ν_i per record. The cost of moving the data from node i to its neighbor node j in the network is μ_{ij} per record. Let, (1) x_{ij} be the amount[4] of data D_i transferred from node i to node j for processing i.e. $0 \leq x_{ij} \leq D_i$. (2) $X = [x_{ij}]_{i,j=1}^{n}$ be the matrix containing all the data transfers in the network and is referred to as a *strategy* [1]. (3) δ_i be the amount of data that can be processed by the i^{th} compute node at the current time t. Note that as more and more jobs are processed, the value of δ_i changes and is always less than the total amount of data that can be processed at a node. The overall cost function for building the data mining model for a strategy X can be obtained as follows:

$$C(X) = \sum_{ij} \mu_{ij} x_{ij} + \nu_j x_{ij} = \sum_{ij} c_{ij} x_{ij} \qquad (12)$$

where $c_{ij} = \mu_{ij} + \nu_j$. The constraints for the optimization problem described by the above equation are derived from the fact that the compute element of each node can process at most δ_i amount of data. Next, we illustrate our optimization problem with an example:

[2]An interested reader is referred to [3] for a detailed survey.

[3]Note that data may be either homogeneously or heterogeneously partitioned and this does not affect our analysis.

[4]This can be expressed as a percentage or total number of records transferred

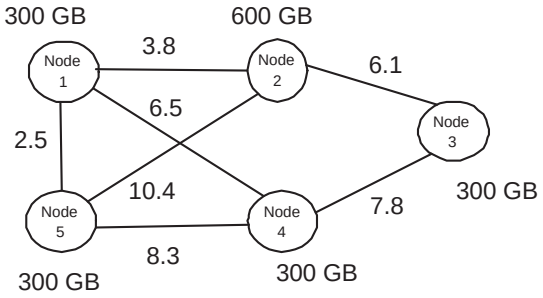


Figure 1. An example of data and bandwidth distribution in a DDM application

Node	ν
1	1.23
2	2.23
3	2.94
4	1.78
5	4.02

Table 2. ν values for the network shown on Figure 1

Example 1 *Consider the network shown in the Figure 1. The weights on the edges represent the cost of moving the data from node i to its neighbor node j i.e. μ_{ij} dollars per record. The values of δ_i for each node are also indicated in the figure, for e.g. $\delta_1 = 300$, $\delta_2 = 600$ etc. Assume that the ν values for each node are as indicated in the Table 2. Thus for the Figure 1 the following objective function can be written:*

$$z = 6.03x_{12} + 9.04x_{23} + 6.52x_{15} + 8.28x_{14} + 14.42x_{25} + 9.58x_{34} + 12.32x_{45}$$

where z is an user defined cost. The corresponding constraints for this objective function are as follows: $x_{12} + x_{14} + x_{15} \leq 300$, $x_{12} + x_{25} + x_{23} \leq 600$, $x_{15} + x_{25} + x_{45} \leq 300$, $x_{14} + x_{34} + x_{45} \leq 300$, $x_{23} + x_{34} \leq 300$, $0 \leq x_{12} \leq D_1$, $0 \leq x_{23} \leq D_2$, $0 \leq x_{15} \leq D_1$, $0 \leq x_{14} \leq D_1$, $0 \leq x_{25} \leq D_2$, $0 \leq x_{34} \leq D_3$, $0 \leq x_{45} \leq D_4$ *Note that in addition to the above mentioned constraints, nodes may have other local constraints developed amongst themselves. For e.g. Nodes 1, 2 and 5 may agree that amount of data transferred from Node 5 to Node 2 is twice sum of data transferred from Node 1 to 2*

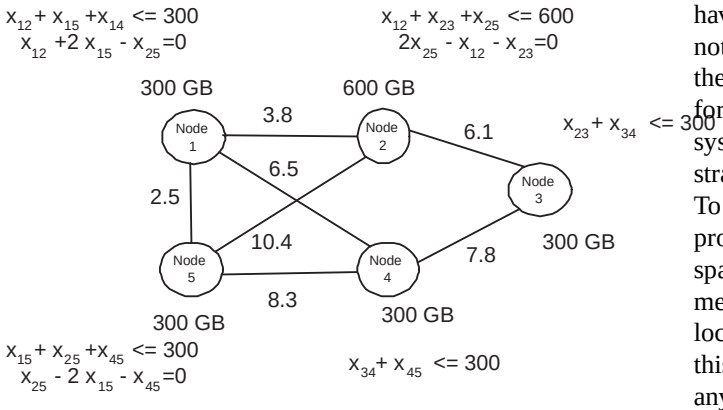


Figure 2. The Distributed LP Problem

and from Node 1 to 5. This constraint may be written as $x_{25} = 2x_{12} + 2x_{15}$.

If the linear programming problem described above is fully constrained it can be solved by direct or iterative techniques. However, if the system is *under constrained*, the solution approaches are different. Historically simplex optimization [2] has been used to solve under constrained linear systems.

5 Distributed Simplex Algorithm

In this section, we describe the distributed simplex algorithm. We first give a description of a method for obtaining the canonical representation of a linear system and then provide details of the algorithm.

5.1 Distributed Canonical Representation of the Linear System

Consider the network described in Example 1. Instead of all the constraints being seen at a centralized site, let Node 1 observe only $x_{12} + x_{14} + x_{15} \leq 300$, Node 2: $x_{12} + x_{23} + x_{25} \leq 600$, Node 3: $x_{23} + x_{34} \leq 300$, Node 4: $x_{34} + x_{45} \leq 300$, Node 5: $x_{15} + x_{25} + x_{45} \leq 300$. In addition let us assume that the following three local constraints are also observed at Nodes 1, 2 and 5 respectively: $x_{12}+2x_{15} = x_{25}$, $2x_{25} = x_{12}+x_{23}$ and $x_{25}-2x_{15}-x_{45} = 0$. Each site still observes the entire objective function, namely: $z = 6.03x_{12} + 9.04x_{23} + 6.52x_{15} + 8.28x_{14} + 14.42x_{25} + 9.58x_{34} + 12.32x_{45}$. Figure 2 illustrates the problem. If all the constraints could have been centralized, the canonical representation would be relatively simple to obtain.

Our objective is to obtain the portion of the tableau corresponding to the local constraints at each node, without having to centralize all the constraints. In order to do so, note that each node in the network needs to have access to the number of basic variables that it should add. This information depends on the total number of constraints in the system. So each node needs to know exactly how many constraints are there in the system at any given point of time. To do so, we propose the following converge cast based approach: Let s be an initiator node which builds a minimum spanning tree on all nodes in the network. Following this, a message is sent by s to all its neighbors asking how many local constraints each node has. A neighbor on receiving this message, either forwards it to its neighbors (if there are any) or sends back a reply. At the end of this procedure, Node s has the correct value of the total number of constraints in the system, say T_c.

Next, Node s sets a variable $count - Constraint$ to the number of its local constraints. It traverses the minimum spanning tree and informs each node visited of the number of constraints seen so far. Let T represent the value of $count - Constraint$ at node i. Then node i must add T_c basic variables to each of its constraints. At the end of this procedure, all nodes have added the relevant basic variables. Note that this procedure creates exactly the same canonical form as would have been obtained if all the constraints were centralized. It must be noted that the Distributed Canonical Representation algorithm needs to be run only once at the time of initialization. Thereafter, each node just updates its tableau depending on the pivots chosen at that round of iteration.

Once each of the nodes have the canonical representation, we are ready to describe the distributed simplex optimization algorithm. We assume that nodes maintain only their local simplex tableau and the global objective function. The goal is to obtain a solution to the global optimization problem.

5.2 Notation and Preliminaries

Let $P_1, P_2, \cdots P_\eta$ be a set of nodes connected to one another via an underlying communication tree such that each node P_i knows its neighbors N_i. Each node P_i has its own local constraints which may change from time to time depending on the resources available at that node. The constraints at node i have the form $A^iX^i = b^i$ where A^i represents an $m \times n$ matrix, X^i is a $n \times 1$ vector and b^i is a $m \times 1$ vector. Thus at each node, we are interested in solving the following linear programming problem: Find $X^i \geq 0$ and Min z^i satisfying $c_1x_1 + c_2x_2 + \cdots c_nx_n = z^i$ subject to the constraints $A^iX^i = b^i$. The global linear program (if all the constraint matrices could be centralized) can be written as follows: Find $X \geq 0$ and Min z satisfying $c_1x_1 + c_2x_2 + \cdots c_nx_n = z$ subject to constraints $AX = B$ where $A = \bigcup_{i=1}^{\eta} A^i$ and $B = \bigcup_{i=1}^{\eta} b^i$.

Local Algorithm: Before describing our algorithm, we first define what we mean by α-neighborhood of a vertex, α-local query and (α, γ)-local algorithm. The α-neighborhood of a node $u \in V$ is defined as, the collection of vertices at a distance α or less from it in G i.e. $\Gamma_\alpha(u, V) = \{v | \text{dist}_G(u, v) \leq \alpha\}$ where $\text{dist}_G(u, v)$ denotes the length of the shortest path in between u and v and the length of a path is measured by the number of edges in it. An α-local query by some vertex v is a query whose response can be computed using some function $f(X_\alpha(v))$ where $X_\alpha(v) = \{X_v | v \in \Gamma_\alpha(v, V)\}$. An algorithm is called (α, γ)-local if it never requires computation of a β-local query such that $\beta > \alpha$ and the total size of the response to all such α-local queries sent out by a peer is bounded by γ. α can be a constant or a function parameterized by the size of the network while γ can be parameterized by both the size of the network and the size of data at a node. Local algorithms can be broadly classified under two categories: (1) Exact local algorithms – Local algorithms that offer the same result that a centralized algorithm will produce. (2) Approximate local algorithms – Local algorithms which rely on the properties of the underlying approximation technique (*e.g.* sampling) to provide error guarantees on the final result and offer approximations of results.

Next, we present an exact local algorithm for solving linear optimization using the simplex method. Our assumption is that each node contains different sets of constraints, but has knowledge of the global objective function.

5.3 The Algorithm

At the beginning of iteration l, a node P_i has its own constraint matrix and the objective function. The column pivot, henceforth referred to as $col - pivot^i$, is that column of the tableau corresponding to the most negative indicator of $c_1, c_2, \cdots, c_n$ (Note that if no negative indicator is found, then this is the final simplex tableau). Following this, each node forms the row ratios ($r^i_j, 1 \leq j \leq m$) for each row i.e. it divides $b^i_j, 1 \leq j \leq m$ by the corresponding number in the pivot column of that row. Let minimum of r^i_j's be presented as $row - pivot^i$. This is stored in the history table of node P_i corresponding to iteration l.

Now the node must participate in a "local" algorithm for determination of the minimum row ratio i.e.

$$\text{Minimum}\,(row - pivot^i), i \in N_i \tag{13}$$

We describe a simple protocol called *Push-Min* for computing equation . At all times t, each node maintains a minimum $m_{t,i}$. At time t=0, $m_{t,i} = row - pivot^i$. Thereafter, each node follows the protocol given in Algorithm 5.3.1. When the protocol Push-Min terminates, each node will have the exact value of the minimum $row - pivot^i$ in the grid. This has been shown by Bawa *et. al.* [8] and we merely state the theorem here for convenience of the reader.

Theorem 1 *Let $\hat{D}$ be the upper bound to the diameter of the network. Assume that Push-Min with value $\hat{D}$ is used by the querying node s for computing the MIN aggregate. The answer of the query would be correct over all nodes i that are constantly connected with s over a path of length at most $\hat{D}$.*

■

Algorithm 5.3.1 Protocol Push-Min

1. Let $\{\hat{m}_r\}$ be all the values sent to i at round $t-1$.
2. Let $m_{t,i} = \min(\{\hat{m}_r\}, row - pivot^i)$
3. Send $m_{t,i}$ to all the neighbors.
4. $m_{t,i}$ is the estimate of the minimum in step t

Once the Push-Min protocol converges, the node containing the minimum $row - pivot^i$ (say P_{min}) will send its row in the simplex tableau to all other nodes in the network. Next node P_i updates its local tableau with respect to the extra row it received from node P_{min}. The algorithm, Local Constraint Sharing Protocol is described in Table 5.3.2. Completion of one round of the LCS-Protocol, ensures that one iteration of the distributed local simplex algorithm is over.

Algorithm 5.3.2 Local Constraint Sharing Protocol (LCS-Protocol)

1. Node P_i performs protocol Push-Min until there are no more messages passed.
2. On convergence to the exact minimum, the minimum $row - pivot^i$ is known to all nodes in the grid.
3. All the nodes use the row obtained in Step 2 to perform Gauss Jordan elimination on the local tableau.
4. At the end of Step 3, each node locally has the updated tableau and completes the current iteration of the simplex algorithm.

Termination: In a termination state, two things should happen: (1) No more messages traverse in the network (2) Each local node has all its $c_i > 0$. Thus the state of the network can be described by information possessed by each node. In particular, each node will have a solution to the linear programming problem and this will be stored in X^i. Note that this solution converges exactly to the solution if all the constraints were centralized.

5.4 Observations

- The algorithm described above, will provide an exact solution to the optimization problem as would have happened if all constraints could have been centralized.

- It remains to be seen, via theoretical analysis and experimentation, how this resource management affects the distributed data mining algorithm. This is beyond the scope of this research.

- If the distributed system were dynamic with nodes joining and leaving on an ad-hoc basis, the constraint matrix for nodes currently in the network will change and so will the objective function to be solved. This significantly complicates the problem and we plan to investigate this in future.

5.5 Analysis of Protocol Push-Min

The Protocol Push-Min behaves similar to the spread of an epidemic in a large population. Consequently our analysis is based on statistical modeling of epidemics ([6, 7]).

Definitions: A node is called *susceptible*, if it does not have the exact minimum, but is capable of obtaining it by communication with its immediate neighbors. If a node receives a $row - pivot$ value less than its current value, it becomes *infected* and willing to share this information with other neighbors. When a node unnecessarily contacts another node which also has the same information, there is no extra information gained by this communication. The node already having the information is called a *dead* or *immune* node. Let x_t represent the number of susceptible nodes, y_t the number of infected ones and z_t the dead or immune nodes. Then,

$$x_t + y_t + z_t = \eta \tag{14}$$

Let β be the *infection* parameter defined as the proportion of contacts between infective and susceptible per unit time; γ is the *removal* parameter defined as the proportion of infective per unit time removed from the population. We also define ρ to be the *Relative Removal Rate*, i.e. $\rho = \frac{\gamma}{\beta}$. Assuming discrete time model, the spread of the minimum value amongst nodes can be represented by the following difference equations:

$$x_{t+1} = x_t - \beta x_t y_t \tag{15}$$

$$y_{t+1} = y_t + \beta x_t y_t - \gamma y_t \tag{16}$$

$$z_{t+1} = z_t + \gamma y_t \tag{17}$$

Next, we illustrate the fact that under the Protocol Push-Min the entire network is infected exponentially fast. This also implies that on convergence, all nodes in the network have the same minimal value.

Lemma 1 *Under the Protocol Push-Min, the number of susceptible nodes in the network decreases exponentially.*

Proof:

Let x_t and x_0 represent the number of susceptible nodes in the network at time t and $t = 0$ respectively. From equation 15 we have,

$$\begin{aligned} \frac{x_{t+1}}{x_t} &= 1 - \beta y_t \\ &= 1 - \frac{z_{t+1} - z_t}{\rho} \text{ (Using equation 17)} \end{aligned}$$

Thus the recursive equation for x_t can be written as follows:,

$$\begin{aligned} x_t &= x_0 \prod_{j=0}^{t-1} 1 - \frac{z_{j+1} - z_j}{\rho} \\ \frac{x_t}{x_0} &= (1 - \frac{z_1 - z_0}{\rho})(1 - \frac{z_2 - z_1}{\rho}) \cdots (1 - \frac{z_t - z_{t-1}}{\rho}) \end{aligned}$$

Now,

$$\lg \frac{x_t}{x_0} = \lg(1 - \frac{z_1 - z_0}{\rho}) + \lg(1 - \frac{z_2 - z_1}{\rho}) + \cdots + \lg(1 - \frac{z_t - z_{t-1}}{\rho})$$

$$\leq (-\frac{z_1 - z_0}{\rho}) + (-\frac{z_2 - z_1}{\rho}) + \cdots + (-\frac{z_t - z_{t-1}}{\rho}) \tag{18}$$

$$\leq -\sum_{j=0}^{t-1} \frac{z_{j+1} - z_j}{\rho}$$

$$\text{Hence, } x_t = x_0 \exp(-\sum_{j=0}^{t-1} \frac{z_{j+1} - z_j}{\rho})$$

Q.E.D.
■

5.6 Convergence of the Local Simplex Algorithm

We have seen in the simplex algorithm described in section 3, the canonical form provides an immediate criteria for testing the optimality of a basic feasible solution. If the criterion is not satisfied, another iteration of the simplex algorithm is initiated. Formally we can state the following theorem:

Theorem 2 (Dantzig, 1963) *Given a linear program presented in feasible canonical form, there exists a finite sequence of pivot operations each yielding a basic feasible*

solution such that the final canonical form yields an optimal basic feasible solution, or an infinite class of feasible solutions for which the values of z have no lower bound.

The proof of Theorem 2 is given in Chapter 6 of [2].
■

In order to prove that the *local* algorithm indeed converges, we prove the following theorem:

Theorem 3 *Assume that the linear constraints at each node can be centralized and a feasible canonical form can be generated. If there exists a finite sequence of pivot operations each yielding a basic feasible solution such that the final canonical form yields an optimal basic feasible solution for the centralized scenario, then such a finite sequence of pivot operations also exists for the Distributed Local algorithm.*

Proof: First note that if the linear constraints at each node were centralized and the objective function was solved using these constraints, the Simplex Algorithm would terminate (Theorem 2). In the distributed scenario, the first step is to generate a canonical representation of the constraints at each node. This is done using the algorithm described in section 5.1. Note that on completion of this initiation step, both the centralized and the local algorithms have the exact same canonical representations. The local algorithm now obtains the column pivot and row pivot. The row pivot obtained after the Protocol Push-Min is executed yields the minimum in the entire network. Thus this is identical to the row pivot that would be obtained in each iteration of the centralized simplex algorithm. This implies that in each iteration, the same row pivot and column pivot are seen both in the centralized and local algorithms. This completes the proof.
■

5.7 Communication Cost Analysis

The communication cost of the local distributed algorithm comes from two parts: (1) The amount of communication required by the Protocol Push-Min until it converges (2) The number of iterations for the simplex algorithm to terminate. In the worst case, Protocol Push-Min may need to communicate with all the nodes in the peer-to-peer grid. This means that worst case communication cost for Push-Min in an iteration of simplex is O (η) (where η is the total number of nodes in the network). Also, it was discussed in section 3.1 that in the worst case, the number of pivots needed by the simplex algorithm is $\binom{n}{m}$. Thus, in the worst case, the communication cost of this distributed algorithm can be exponential. However, in most practical cases, the simplex algorithm converges in λm [2] iterations where $\lambda < 4$ typically. This means that for most practical cases, communication complexity is at most O ($\lambda m \eta$). Note that centralization of data would require O (m n) communication. Thus if $\eta < \frac{n}{\lambda}$ significant benefits may be obtained from this distributed resource management algorithm.

6 Experimental Results

This section presents experimental results for the local algorithm for simplex optimization. We describe the dataset, and the performance of our algorithm.

6.1 Dataset

The dataset at each node comprises of the A^i and the b^i matrices. We simulated both these matrices and added basic variables depending on the total number of constraints in the network as described in section 5.1. Different nodes are allowed to have different number of constraints but always have the same number of variables i.e. $x_1, x_2, \cdots x_n$. The goal at each node was to always solve the same objective function i.e. Minimize $c_1x_1 + c_2x_2 + \cdots c_nx_n$. The centralized dataset was obtained by concatenating all the constraints over all nodes in the network and the corresponding values in the b^i matrix.

6.2 Performance

We measure the communication cost of our algorithm as follows: Let τ_h be the total number of messages passed in execution of Protocol Push-Min in the h^{th} iteration of the algorithm. Let K be the total number of iterations required by the local simplex algorithm. Then Total Communication Cost (TCC) is given as follows:

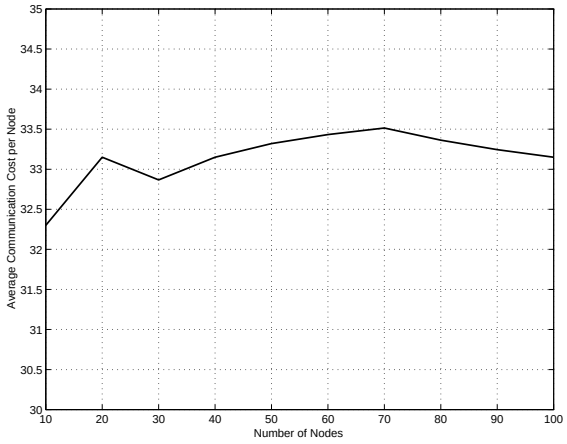


Figure 3. Average Communication Cost Per Node versus Number of Nodes.

$$TCC = \sum_{h=1}^{K} \tau_h \quad (19)$$

We vary the number of nodes in the network, keeping the size of the constraint matrix (m) and number of variables (n) fixed. We report the Average Communication Cost per Node (ACCN) as follows:

$$ACCN = \frac{TCC}{\eta} \quad (20)$$

For our experiments the size of the simplex tableau was chosen to be 101×301. This means that $m = 100$ and $n = 300$. The constraint matrix was evenly split amongst different number of nodes in the network. For example when there are 10 nodes, each node had a constraint matrix of size 10×300 and so on. The number of messages passed in each iteration of the simplex for computing the minimum row pivot was noted and summed up over all iterations until the simplex algorithm terminated. The Figure 3 illustrates our results. The more or less flat nature of the curve indicates that the local algorithm scales well i.e. as the number of nodes in the network increases, the average communication cost per node still remains more or less constant.

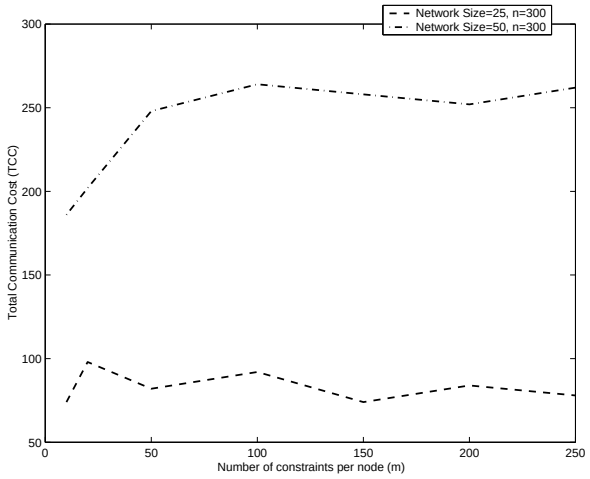


Figure 4. Total Communication Cost (TCC) versus Number of Constraints at each node

We also performed experiments to test the scalability of the local algorithm with regard to: (1) The number of constraints (m) and (2) The number of variables (n). The Figure 4 shows the variation of the total communication cost versus the number of constraints at each node. For this experiment, we used two different graph topologies containing 25 and 50 nodes each. The number of variables used was 300 at each node. In Figure 5 we examine the variation of the total communication cost versus the number of variables at

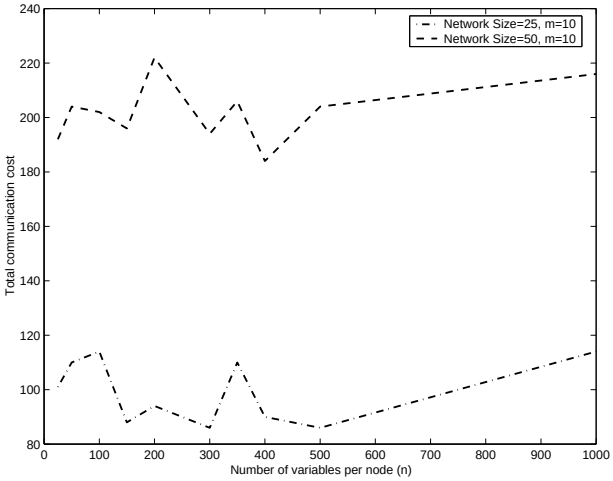


Figure 5. Total Communication Cost (TCC) versus Number of Variables at each node

each node in the network. The network size was 25 and 50 as before, but the number of constraints at each node was kept constant at 10. Interestingly, it must be noted that the communication occurs with the neighbors only when a minimum $row - pivot$ value in the entire network has to be determined. Note that all other pivoting operations are done locally at a node. This provides a more or less flat curve even when the number of variables at a node is as large as 1000.

7 Conclusions and Future Work

Applications storing large quantities of distributed data (such as astronomy sky surveys, telemedicine, e-commerce applications) are becoming very popular in recent years. Management of resources and data on these decentralized applications is a challenging problem due to the inherent hetrogeneity in the data, bandwidth limitations and access control restrictions imposed by the applications. In this paper, we design an algorithm for distributed resource and data management using the well known simplex algorithm. We present extensive analytical and empirical results to demonstrate the performance of the algorithm. To the best of our knowledge, this is the first work that emphasizes the need for developing large scale distributed optimization algorithms to enhance resource management and mining on distributed systems. There are several directions for future work such as developing algorithms for *dynamic* distributed mining applications, studying non-linear optimization techniques and different formulations of optimization problems such as relaxing the requirement that all nodes have knowledge of the objective function.

8 Acknowledgements

Removed for double blind review.

References

[1] Turinsky, A. L, *Balancing Cost and Accuracy in Distributed Data Mining*, PhD Thesis, University of Illinois at Chicago, 2002.

[2] Dantzig, G. B, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

[3] Shamir, R, *The efficiency of the simplex method: a survey*, Management Science, Vol. 33, No. 3, 1987.

[4] Dantzig, G. B, *Comments on Khachian's Algorithm for Linear Programming*, Tech. Report, Stanford University, SOL-79-22, 1979.

[5] Klee, V. and Minty, G. J, *How Good is the Simplex Algorithm?*, Inequalities III, Pg 159175, New York, 1972.

[6] Demers, A. and Greene, D. and Hauser, C. and Irish, W. and Larson, J. and Shenker, S. and Sturgis, H. and Swinehart, D. and Terry, D, *Epidemic Algorithms for Replicated Database Maintenance*, Proceedings of the sixth annual ACM Symposium on Principles of distributed computing, Pgs 1- 12, New York, 1987.

[7] Pittel, B, *On spreading a rumor*, SIAM J. Appl. Math., Vol. 47, No. 1, Philadelphia, PA, 1987.

[8] Mayank B., Hector G. M., Aristides, G. and Motwani R. *Estimating Aggregates on a Peer-to-Peer Network*, Tech. Report, Stanford University, 2004.

[9] Iamnitchi A, *Resource Discovery in Large Resource-Sharing Environments*, Ph.D. Thesis, University of Chicago, 2003.

[10] Raman R, Livny M. and Solomon M, *Matchmaking: Distributed resource management for high throughput computing*, The 7th IEEE International Symposium on High Performance Distributed Computing, Pages 28-31, 1998.

[11] Awan A, Ferreira R. A, Jagannathan S. and Grama A. *Unstructured Peer-to-Peer Networks for Sharing Processor Cycles*, Parallel Computing, Vol 32, 2006.

[12] Forsgren A, Gill P. E. and Wright M. H. *Interior Methods for Nonlinear Optimization*, SIAM Review, Vol 44, Pgs 525-597, 2002.

[13] Karmarkar N, *A New Polynomial-Time Algorithm for Linear Programming*, Combinatorica, Vol 4, Pgs 373-395, 1984.

[14] Yarmish G, *A Distributed Implementation of the Simplex Method*, Ph.D. Thesis, Computer and Information Science, Polytechnic University, 2001.

[15] Hall, J. A. J. and McKinnon, K. I. M. *Update procedures for the parallel revised simplex method*, Tech. report, University of Edinburgh, U.K, 1992.

[16] Craig, S. and Reed, D. *Hypercube Implementation of the Simplex Algorithm*, Association of Computing Machinery (ACM), pgs 1473 - 1482, 1988

[17] Ho, J. K. and Sundarraj, R. P. *On the efficacy of distributed simplex algorithms for linear programming*, Comput. Optim. Appl., Vol 3, No 4, Pgs 349 - 363, 1994.

[18] Papadimitriou, C. H. and Yannakakis, M. *Linear programming without the matrix*, STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, Pgs 121–129, San Diego, CA, 1993.

[19] Jonathan E, Boduroglu I, Polymenakos L and Goldfarb, D. *Data-Parallel Implementations of Dense Simplex Methods on the Connection Machines CM-2*, ORSA Journal on Computing (INFORMS), Vol 7, No. 4, Pgs 402-416.

[20] Bartal Y, Byers J. W and Raz D. *Fast, Distributed Approximation Algorithms for Positive Linear Programming with Applications to Flow Control*, SIAM J. Computing, Vol 33, No. 6, Pgs 1261-1279, Philadelphia, PA, 2004.

[21] H. Kargupta and P. Chan. (editors) *Advances in Distributed and Paral lel Knowledge Discovery*. AAAI press, Menlo Park, CA, 2000.

[22] M. Zaki. *Parallel and Distributed Data Mining: An Introduction*. In Large-Scale Paral lel Data Mining (Lecture Notes in Articial Intel ligence 1759), edited by Zaki M. and Ho C.-T., Springer-Verlag, Berlin, pages 123, 2000.

[23] M. A. Goberna and M. A. Lopez, *Linear Semi-Innite Optimization*, John Wiley, New York, 1998.

[24] G. Fung, O. L. Mangasarian, and J. Shavlik. *Knowledge-based nonlinear kernel classiers.*, Technical Report 03-02, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, March 2003.

A Generic Local Algorithm for Mining Data Streams in Large Distributed Systems

Ran Wolff, Kanishka Bhaduri, and Hillol Kargupta *Senior Member, IEEE*

***Abstract*— In a large network of computers or wireless sensors, each of the components (henceforth, peers) has some data about the global state of the system. Much of the system's functionality such as message routing, information retrieval and load sharing relies on modeling the global state. We refer to the outcome of the function (*e.g.*, the load experienced by each peer) as the *model* of the system. Since the state of the system is constantly changing, it is necessary to keep the models up-to-date.**

Computing global data mining models *e.g.* decision trees, k-means clustering in large distributed systems may be very costly due to the scale of the system and due to communication cost, which may be high. The cost further increases in a dynamic scenario when the data changes rapidly. In this paper we describe a two step approach for dealing with these costs. First, we describe a highly efficient *local* algorithm which can be used to monitor a wide class of data mining models. Then, we use this algorithm as a feedback loop for the monitoring of complex functions of the data such as its k-means clustering. The theoretical claims are corroborated with a thorough experimental analysis.

I. Introduction

In sensor networks, peer-to-peer systems, grid systems, and other large distributed systems there is often the need to model the data that is distributed over the entire system. In most cases, centralizing all or some of the data is a costly approach. When data is streaming and system changes are frequent, designers face a dilemma: should they update the model frequently and risk wasting resources on insignificant changes, or update it infrequently and risk model inaccuracy and the resulting system degradation.

At least three algorithmic approaches can be followed in order to address this dilemma: The *periodic* approach is to rebuild the model from time to time. The *incremental* approach is to update the model with every change of the data. Last, the *reactive* approach, what we describe here, is to monitor the change and rebuild the model only when it no longer suits the data. The benefit of the periodic approach is its simplicity and its fixed costs in terms of communication and computation. However, the costs are fixed independent of the fact that the data is static or rapidly changing. In the former case the periodic approach wastes resources, while on the latter it might be inaccurate. The benefit of the incremental approach is that its accuracy can be optimal. Unfortunately, coming up with incremental algorithms which are both accurate and efficient can be hard and problem specific. On the other hand, model accuracy is usually judged according to a small number of rather simple metrics (misclassification error, least square error, etc.). If monitoring is done efficiently and accurately, then the reactive approach can be applied to many different data mining algorithm at low costs.

Local algorithms are one of the most efficient family of algorithms developed for distributed systems. Local algorithms are in-network algorithms in which data is never centralized but rather computation is performed by the peers of the network. At the heart of a local algorithm there is a data dependent criteria dictating when nodes can avoid sending updates to their neighbors. An algorithm is generally called local if this criteria is independent with respect to the number of nodes in the network. Therefore, in a local algorithm, it often happens that the overhead is independent of the size of the system. Primarily for this reason, local algorithms exhibit high scalability. The dependence on the criteria for avoiding to send messages also makes local algorithms inherently incremental. Specifically, if the data changes in a way that does not violate the criteria, then the algorithm adjusts to the change without sending any message.

Local algorithms were developed, in recent years, for a large selection of data modeling problems. These include association rule mining [1], facility location [2], outlier detection [3], L2 norm monitoring [4], classification [5], and multivariate regression [6]. In all these cases, resource consumption was shown to converge to a constant when the number of nodes is increased. Still, the main problem with local algorithms, thus far, has been the need to develop one for every specific problem.

In this work we make the following progress. First, we generalize a common theorem underlying the local algorithms in [1], [2], [4], [5], [6] extending it from $\mathbb{R}$ to $\mathbb{R}^d$. Next, we describe a generic algorithm, relying on the said generalized theorem, which can be used to compute arbitrarily complex functions of the average of the data in a distributed system; we show how the said algorithm can be extended to other linear combinations of data, including weighted averages of selections from the data. Then, we describe a general framework for monitoring, and consequent reactive updating of any model of horizontally distributed data. Finally, we describe the application of this framework for the problem of providing a

A preliminary version of this work was published in the Proceedings of the 2006 SIAM Data Mining Conference (SDM'06).

Manuscript received ...; revised ....

Ran Wolff is with the Department of Management Information Systems, Haifa University, Haifa-31905, Israel. Email:rwolff@mis.haifa.il. Kanishka Bhaduri is with Mission Critical Technologies Inc at NASA Ames Research Center, Moffett Field CA 94035. Email:kanishka_bh@yahoo.com. Hillol Kargupta is with the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, Baltimore, MD 21250. E-mail:hillol@cs.umbc.edu. Hillol Kargupta is also affiliated to AGNIK LLC, Columbia, MD 21045. This work was done when Kanishka Bhaduri was at UMBC.

k clustering which is a good approximation of the k-means clustering of data distributed over a large distributed system. Our theoretical and algorithmic results are accompanied with a thorough experimental validation, which demonstrates both the low cost and the excellent accuracy of our method.

The rest of this paper is organized as follows. The next section describes our notations, assumptions, and the formal problem definition. In Section III we describe and prove the main theorem of this paper. Following, Section IV describes the generic algorithm and its specification for the L2 thresholding problem. Section V presents the reactive algorithms for monitoring three typical data mining problems – *viz.* means monitoring and k-means monitoring. Experimental evaluation is presented in Section VI while Section VII describes related work. Finally, Section VIII concludes the paper and lists some prospective future work.

II. Notations, Assumptions, and Problem Definition

In this section we discuss the notations and assumptions which will be used throughout the rest of the paper. The main idea of the algorithm is to have peers accumulate sets of input vectors (or summaries thereof) from their neighbors. We show that under certain conditions on the accumulated vectors a peer can stop sending vectors to its neighbors long before it collects all input vectors. Under these conditions one of two things happens: Either all peers can compute the result from the input vectors they have already accumulated or at least one peer will continue to update its neighbors – and through them the entire network – until all peers compute the correct result.

A. Notations

Let $V = \{p_1, \ldots, p_n\}$ be a set of peers (we use the term peers to describe the peers of a peer-to-peer system, motes of a wireless sensor network, etc.) connected to one another via an underlying communication infrastructure. The set of peers with which p_i can directly communicate, N_i, is known to p_i. Assuming connectedness, N_i always contains p_i and at least one more peer. Additionally, p_i is given a time varying set of input vectors in $\mathbb{R}^d$.

Peers communicate with one another by sending sets of input vectors (below, we show that for our purposes statistics on sets are sufficient). We denote by $X_{i,j}$ the latest set of vectors sent by peer p_i to p_j. For ease of notation, we denote the input of p_i (mentioned above) $X_{i,i}$. Thus, $\bigcup_{p_j \in N_i} X_{j,i}$ becomes the latest set of input vectors known to p_i.

Assuming reliable messaging, once a message is delivered both p_i and p_j know both $X_{i,j}$ and $X_{j,i}$. We further define four sets of vectors that are central to our algorithm.

Definition 2.1: The *knowledge* of p_i, is $\mathcal{K}_i = \bigcup_{p_j \in N_i} X_{j,i}$.

Definition 2.2: The *agreement* of p_i and any neighbor $p_j \in N_i$ is $\mathcal{A}_{i,j} = X_{i,j} \cup X_{j,i}$.

Definition 2.3: The *withheld knowledge* of p_i with respect to a neighbor p_j is the subtraction of the agreement from the knowledge $\mathcal{W}_{i,j} = \mathcal{K}_i \setminus \mathcal{A}_{i,j}$.

Definition 2.4: The *global input* is the set of all inputs $\mathcal{G} = \bigcup_{p_i \in V} X_{i,i}$.

We are interested in inducing functions defined on $\mathcal{G}$. Since $\mathcal{G}$ is not available at any peer, we derive conditions on $\mathcal{K}$, $\mathcal{A}$ and $\mathcal{W}$ which will allow us to learn the function on $\mathcal{G}$. Our next set of definitions deal with convex regions which are a central point of our main theorem to be discussed in the next section.

A region $R \subseteq \mathbb{R}^d$ is convex, if for every two points $x, y \in R$ and every $\alpha \in [0, 1]$, the weighted average $\alpha \cdot x + (1 - \alpha) \cdot y \in R$. Let $\mathcal{F}$ be a function from $\mathbb{R}^d$ to an arbitrary domain $\mathbb{O}$. $\mathcal{F}$ is constant on R if $\forall x, y \in R : \mathcal{F}(x) = \mathcal{F}(y)$. Any set or regions $\{R_1, R_2, \ldots\}$ induces a cover of $\mathbb{R}^d$, $\mathcal{R} = \{R_1, R_2, \ldots, T\}$ in which the *tie* region T includes any point of $\mathbb{R}^d$ which is not included by one of the other regions. We denote a given cover $\mathcal{R}_\mathcal{F}$ *respective* of $\mathcal{F}$ if for all regions except the tie region $\mathcal{F}$ is constant. Finally, for any $x \in \mathbb{R}^d$ we denote $\mathcal{R}_\mathcal{F}(x)$ the first region of $\mathcal{R}_\mathcal{F}$ which includes x.

B. Assumptions

Throughout this paper, we make the following assumptions:

Assumption 2.1: Communication is reliable.

Assumption 2.2: Communication takes place over a spanning communication tree.

Assumption 2.3: Peers are notified on changes in their own data x_i, and in the set of their neighbors N_i.

Assumption 2.4: Input vectors are unique.

Assumption 2.5: A respective cover $\mathcal{R}_\mathcal{F}$ can be precomputed for $\mathcal{F}$.

Note that assumption 2.1 can easily be enforced in all architectures as the algorithm poses no requirement for ordering or timeliness of messages. Simple approaches, such as piggybacking message acknowledgement can thus be implemented in even the most demanding scenarios – those of wireless sensor networks. Assumption 2.3 can be enforced using a heartbeat mechanism. Assumption 2.2 is the strongest of the three. Although solutions that enforce it exist (see for example [7]), it seems a better solution would be to remove it altogether using a method as described by Liss *et al.* [8]. However, describing such a method in this generic setting is beyond the scope of this paper. Assumption 2.4 can be enforced by adding the place and time of origin to each point and then ignoring it in the calculation of $\mathcal{F}$. Assumption 2.5 does not hold for any function. However, it does hold for many interesting ones. The algorithm described here can be sensitive to an inefficient choice of respective cover.

Note that, the correctness of the algorithm cannot be guaranteed in case the assumptions above do not hold. Specifically, duplicate counting of input vectors can occur if Assumption 2.2 does not hold — leading to any kind of result. If messages are lost then not even consensus can be guaranteed. The only positive result which can be proved quite easily is that if at any time the communication infrastructure becomes a forest, any tree will converge to the value of the function on the input of the peers belonging to that tree.

C. Sufficient statistics

The algorithm we describe in this paper deals with computing functions of linear combinations of vectors in $\mathcal{G}$. For clarity, we will focus on one such combination – the average. Linear combinations, and the average among them, can be computed from statistics. If each peer learns any input vector (other than its own) through just one of its neighbors, then for the purpose of computing $\mathcal{K}_i$, $\mathcal{A}_{i,j}$, and $\mathcal{W}_{i,j}$, the various $X_{i,j}$ can be replaced with their average, $\overline{X_{i,j}}$, and their size, $|X_{i,j}|$. To make sure that happens, all that is required from the algorithm is that the content of every message sent by p_i to its neighbor p_j would not be dependent on messages p_j previously sent to p_i. In this way, we can rewrite:

- $|\mathcal{K}_i| = \sum_{p_j \in N_i} |X_{j,i}|$
- $|\mathcal{A}_{i,j}| = |X_{i,j}| + |X_{j,i}|$
- $|\mathcal{W}_{i,j}| = |\mathcal{K}_i| - |\mathcal{A}_{i,j}|$
- $\overline{\mathcal{K}_i} = \sum_{p_j \in N_i} \frac{|X_{j,i}|}{|\mathcal{K}_i|} \overline{X_{j,i}}$
- $\overline{\mathcal{A}_{i,j}} = \frac{|X_{i,j}|}{|\mathcal{A}_{i,j}|}\overline{X_{i,j}} + \frac{|X_{j,i}|}{|\mathcal{A}_{i,j}|}\overline{X_{j,i}}$
- $\overline{\mathcal{W}_{i,j}} = \frac{|\mathcal{K}_i|}{|\mathcal{W}_{i,j}|}\overline{\mathcal{K}_i} - \frac{|\mathcal{A}_{i,j}|}{|\mathcal{W}_{i,j}|}\overline{\mathcal{A}_{i,j}}$ or nil in case $|\mathcal{W}_{i,j}| = 0$.

D. Problem Definition

We now formally define the kind of computation provided by our generic algorithm and our notion of correct and of accurate computation.

Problem definition: Given a function $\mathcal{F}$, a spanning network tree $G(V, E)$ which might change with time, and a set of time varying input vectors $X_{i,i}$ at every $p_i \in V$, the problem is to compute the value of $\mathcal{F}$ over the average of the input vectors $\overline{\mathcal{G}}$.

While the problem definition is limited to averages of data it can be extended to weighted averages by simulation. If a certain input vector needs to be given an integer weight ω then ω peers can be simulated inside the peer that has that vector and each be given that input vector. Likewise, if it is desired that the average be taken only over those inputs which comply with some selection criteria then each peer can apply that criteria to $X_{i,i}$ apriori and then start off with the filtered data. Thus, the definition is quite conclusive.

Because the problem is defined for data which may change with time, a proper definition of algorithmic correctness must also be provided. We define the *accuracy* of an algorithm as the number of peers which compute the correct result at any given time, and denote an algorithm as *robust* if it presents constant accuracy when faced with stationarily changing data. We denote an algorithm as *eventually correct* if, once the data stops changing, and regardless of previous changes, the algorithm is guaranteed to converge to a hundred percent accuracy.

Finally, the focus of this paper is on *local* algorithms. As defined in [1], a local algorithm is one whose performance is not inherently dependent on the system size, *i.e.*, in which $|V|$ is not a factor in any lower bound on performance. Notice locality of an algorithm can be conditioned on the data. For instance, in [1] a majority voting algorithm is described which may perform as badly as $O\left(|V|^2\right)$ in case the vote is tied. Nevertheless when the vote is significant and the distribution of votes is random the algorithm will only consume constant resources, regardless of $|V|$. Alternative definitions exist for local algorithms and are thoroughly discussed in [9] and [10].

III. Main Theorems

The main theorem of this paper lay the background for a local algorithm which guarantees eventual correctness in the computation of a wide range of ordinal functions. The theorem generalizes the local stopping rule described in [1] by describing a condition which bounds the whereabouts of the global average vector in $\mathbb{R}^d$ depending on the $\mathcal{K}_i$ $\mathcal{A}_{i,j}$ and $\mathcal{W}_{i,j}$ of each peer p_i.

Theorem 3.1: **[Main Theorem]** Let $G(V, E)$ be a spanning tree in which V is a set of peers and let $X_{i,i}$ be the input of p_i, $\mathcal{K}_i$ be its knowledge, and $\mathcal{A}_{i,j}$ and $\mathcal{W}_{i,j}$ be its agreement and withheld knowledge with respect to a neighbor $p_j \in N_i$ as defined in the previous section. Let $R \subseteq \mathbb{R}^d$ be any convex region. If at a given time no messages traverse the network and for all p_i and $p_j \in N_i$ $\overline{\mathcal{K}_i}, \overline{\mathcal{A}_{i,j}} \in R$ and either $\mathcal{W}_{i,j} = \emptyset$ or $\overline{\mathcal{W}_{i,j}} \in R$ as well, then $\overline{\mathcal{G}} \in R$.

Proof: Consider a communication graph $G(V, E)$ in which for some convex R and every p_i and p_j such that $p_j \in N_i$ it holds that $\overline{\mathcal{K}_i}, \overline{\mathcal{A}_{i,j}} \in R$ and either $\mathcal{W}_{i,j} = \emptyset$ or $\overline{\mathcal{W}_{i,j}} \in R$ as well. Assume an arbitrary leaf p_i is eliminated and all of the vectors in $\mathcal{W}_{i,j}$ are added to its sole neighbor p_j. The new knowledge of p_j is $\mathcal{K'}_j = \mathcal{K}_j \cup \mathcal{W}_{i,j}$. Since by definition $\mathcal{K}_j \cap \mathcal{W}_{i,j} = \emptyset$, the average vector of the new knowledge of p_j, $\overline{\mathcal{K'}_j}$, can be rewritten as $\overline{\mathcal{K}_j \cup \mathcal{W}_{i,j}} = \alpha \cdot \overline{\mathcal{K}_j} + (1-\alpha) \cdot \overline{\mathcal{W}_{i,j}}$ for some $\alpha \in [0, 1]$. Since R is convex, it follows from $\overline{\mathcal{K}_j}, \overline{\mathcal{W}_{i,j}} \in R$ that $\overline{\mathcal{K'}_j} \in R$ too.

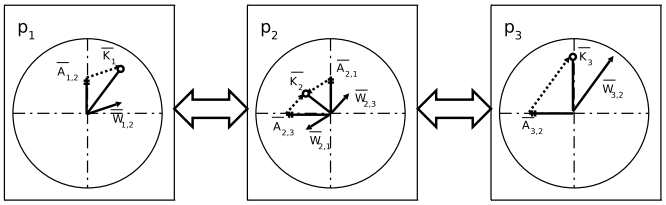
Now, consider the change in the withheld knowledge of p_j with respect to any other neighbor $p_k \in N_j$ resulting from sending such a message. The new $\mathcal{W'}_{j,k} = \mathcal{W}_{i,j} \cup \mathcal{W}_{j,k}$. Again, since $\mathcal{W}_{i,j} \cap \mathcal{W}_{j,k} = \emptyset$ and since R is convex it follows from $\mathcal{W}_{i,j}, \mathcal{W}_{j,k} \in R$ that $\mathcal{W'}_{j,k} \in R$ as well. Finally, notice the agreements of p_j with any neighbor p_k except p_i do not change as a result of such message.

Hence, following elimination of p_i we have a communication tree with one less peer in which the same conditions still apply to every remaining peer and its neighbors. Proceeding with elimination we can reach a tree with just one peer p_1, still assured that $\overline{\mathcal{K}_1} \in R$. Moreover, since no input vector was lost at any step of the elimination $\mathcal{K}_1 = \mathcal{G}$. Thus, we have that under the said conditions $\overline{\mathcal{G}} \in R$. ■

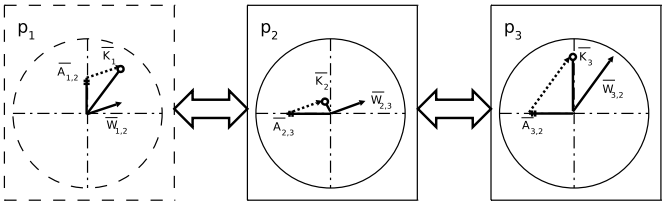
Theorem 3.1 is exemplified in Figure 1. Three peers are shown, each with a drawing of its knowledge, it agreement with its neighbor or neighbors, and the withheld knowledge. Notice the agreement $\overline{\mathcal{A}_{1,2}}$ drawn for p_1 is identical to $\overline{\mathcal{A}_{2,1}}$ at p_2. For graphical simplicity we assume all of the vectors have the same weight – and avoid expressing it. We also depict the withheld knowledge vectors twice – once as a subtraction of the agreement from the knowledge – using a dotted line – and once – shifted to the root – as measured in practice. If the position of the three peers' data is considered vis-a-vis the circular region then the conditions of Theorem 3.1 hold.

Now, assume what would happen when peer p_1 is eliminated. This would mean that all of the knowledge it withholds from p_2 is added to $\mathcal{K}_2$ and to $\mathcal{W}_{2,3}$. Since we assumed $|\mathcal{W}_{1,2}| = |\mathcal{K}_2| = 1$ the result is simply the averaging of the previous $\overline{\mathcal{K}_2}$ and $\overline{\mathcal{W}_{1,2}}$. Notice both these vectors remain in the circular region.

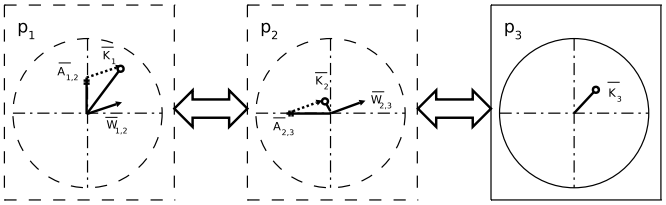
Lastly, as p_2 is eliminated as well, $\mathcal{W}_{2,3}$ – which now also includes $\mathcal{W}_{1,2}$ – is blended into the knowledge of p_3. Thus, $\mathcal{K}_3$ becomes equal to $\mathcal{G}$. However, the same argument, as applied in the elimination of p_1, assures the new $\overline{\mathcal{K}_3}$ is in the circular region as well.



(a) Three peers p_1, p_2 and p_3 where p_2 is connected to both other peers.



(b) After elimination of p_1.



(c) After elimination of p_2.

Fig. 1. At Figure 1(a) the data at all three peers concur with the conditions of Theorem 3.1 with respect to the circle – which is a convex region. If subsequently peer p_1 is eliminated and $\mathcal{W}_{1,2}$ sent to p_2 then $\mathcal{A}_{2,3}$ is not affected and $\mathcal{K}_2$ and $\mathcal{W}_{2,3}$ do change but still remain in the same region. When subsequently, in Figure 1(c), p_2 is eliminated again $\mathcal{K}_3 = \mathcal{G}$ which demonstrates $\overline{\mathcal{G}}$ is in the circular region.

To see the relation of Theorem 3.1 to the previous the Majority-Rule algorithm [1], one can restate the majority voting problem as deciding whether the average of zero-one votes is in the segment $[0, \lambda)$ or the segment $[\lambda, 1]$. Both segments are convex, and the algorithm only stops if for all peers the knowledge is further away from λ than the agreement – which is another way to say the knowledge, the agreement, and the withheld data are all in the same convex region. Therefore, Theorem 3.1 generalizes the basic stopping rule of Majority-Rule to any convex region in $\mathbb{R}^d$.

Two more issues arise from this comparison: one is that in Majority-Rule the regions used by the stopping rule coincide with the regions in which $\mathcal{F}$ is constant. The other is that in the Majority-Rule, every peer decides according to which of the two regions it should try to stop by choosing the region which includes the agreement. Since there are just two non-overlapping region, peers reach consensus on the choice of region and, hence, on the output.

These two issues become more complex for a general $\mathcal{F}$ over $\mathbb{R}^d$. First, for many interesting $\mathcal{F}$, the regions in which the function is constant are not all convex. Also, there could be many more than two such regions, and the selection of the region in which the stopping rule needs be evaluated becomes non-trivial.

We therefore provide two lemmas which provide a way to deal with the selection problem and an answer to the case where in which a function cannot be neatly described as a partitioning of $\mathbb{R}^d$ to convex regions in which it is constant.

Lemma 3.2: **[Consensus]** Let $G(V, E)$ be a spanning tree in which V is a set of peers and let $X_{i,i}$ be the input of p_i, $\mathcal{K}_i$ be its knowledge, and $\mathcal{A}_{i,j}$ and $\mathcal{W}_{i,j}$ be its agreement and withheld knowledge with respect to a neighbor $p_j \in N_i$ as defined in the previous section. Let $\mathcal{R}_\mathcal{F} = \{R_1, R_2, \ldots, T\}$ be a $\mathcal{F}$-respective cover, and let $\mathcal{R}_\mathcal{F}(x)$ be the first region in $\mathcal{R}_\mathcal{F}$ which contains x. If for every peer p_i and every $p_j \in N_i$ $\mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right) = \mathcal{R}_\mathcal{F}\left(\overline{\mathcal{A}_{i,j}}\right)$ then for every two peers p_i and p_ℓ, $\mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right) = \mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_\ell}\right)$.

Proof: We prove this by contradiction. Assume that the result is not true. Then there are two peers p_i and p_ℓ with $\mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right) \neq \mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_\ell}\right)$. Since the communication graph is a spanning tree, there is a path from p_i to p_ℓ and somewhere along that path there are two neighbor peers, p_u and p_v such that $\mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_u}\right) \neq \mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_v}\right)$. Notice, however, that $\overline{\mathcal{A}_{u,v}} = \overline{\mathcal{A}_{v,u}}$. Therefore, either $\mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_u}\right) \neq \mathcal{R}_\mathcal{F}\left(\overline{\mathcal{A}_{u,v}}\right)$ or $\mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_v}\right) \neq \mathcal{R}_\mathcal{F}\left(\overline{\mathcal{A}_{v,u}}\right)$ — a contradiction. ■

Building on Lemma 3.2 above, a variant of Theorem 3.1 can be proved which makes use of a respective cover to compute the value of $\mathcal{F}$.

Theorem 3.3: Let $G(V, E)$ be a spanning tree in which V is a set of peers and let $X_{i,i}$ be the input of p_i, $\mathcal{K}_i$ be its knowledge, and $\mathcal{A}_{i,j}$ and $\mathcal{W}_{i,j}$ be its agreement and withheld knowledge with respect to a neighbor $p_j \in N_i$ as defined in the previous section. Let $\mathcal{R}_\mathcal{F} = \{R_1, R_2, \ldots, T\}$ be a respective cover, and let $\mathcal{R}_\mathcal{F}(x)$ be the first region in $\mathcal{R}_\mathcal{F}$ which contains x. If for every peer p_i and every $p_j \in N_i$ $\mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right) = \mathcal{R}_\mathcal{F}\left(\overline{\mathcal{A}_{i,j}}\right) \neq T$ and if furthermore either $\mathcal{W}_{i,j} = \emptyset$ or $\overline{\mathcal{W}_{i,j}} \in \mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$ then for every p_i, $\mathcal{F}(\overline{\mathcal{K}_i}) = \mathcal{F}(\overline{\mathcal{G}})$.

Proof: From Lemma 3.2 it follows that all peers compute the same $\mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$. Thus, since this region is not T, it must be convex. It therefore follows from Theorem 3.1 that $\overline{\mathcal{G}}$ is, too, in $\mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$. Lastly, since $\mathcal{R}_\mathcal{F}$ is a respective cover $\mathcal{F}$ must be constant on all regions except T. Thus, the value of $\mathcal{F}(\overline{\mathcal{G}})$ is equal to that of $\mathcal{F}(\overline{\mathcal{K}_i})$, for any p_i. ■

IV. A Generic Algorithm and Its Instantiation

This section describes a generic algorithm which relies on the results presented in the previous section to compute the value of a given function of the average of the input vectors. This generic algorithm is both local and eventually correct. The section proceeds to exemplify how this generic algorithm can be used by instantiating it to compute whether the average vector has length above a given threshold $\mathcal{F}(x) =$

$\begin{cases} 0 & \|x\| \leq \epsilon \\ 1 & \|x\| > \epsilon \end{cases}$. L2 thresholding is both an important problem in its own right and can also serve as the basis for data mining algorithms as will be described in the next section.

A. Generic Algorithm

The generic algorithm, depicted in Algorithm 1, receives as input the function $\mathcal{F}$, a respective cover $\mathcal{R}_{\mathcal{F}}$, and a constant, L, whose function is explained below. Each peer p_i outputs, at every given time, the value of $\mathcal{F}$ based on its knowledge $\overline{\mathcal{K}_i}$.

The algorithm is event driven. Events could be one of the following: a message from a neighbor peer, a change in the set of neighbors (*e.g.*, due to failure or recovery), a change in the local data, or the expiry of a timer which is always set to no more than L. On any such event p_i calls the **OnChange** method. When the event is a message $\overline{X}, |X|$ received from a neighbor p_j, p_i would update $\overline{X_{i,j}}$ to $\overline{X}$ and $|X_{i,j}|$ to $|X|$ before it calls **OnChange**.

The objective of the **OnChange** method is to make certain that the conditions of Lemma 3.3 are maintained for the peer that runs it. These conditions require $\overline{\mathcal{K}_i}$, $\overline{\mathcal{A}_{i,j}}$, and $\overline{\mathcal{W}_{i,j}}$ (in case it is not null) to all be in $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$, which is not the tie region T. Of the three, $\overline{\mathcal{K}_i}$ cannot be manipulated by the peer. The peer thus manipulates both $\overline{\mathcal{A}_{i,j}}$, and $\overline{\mathcal{W}_{i,j}}$ by sending a message to p_j, and subsequently updating $X_{i,j}$.

In case $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right) \neq T$ one way to adjust $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$ so that the conditions of Lemma 3.3 are maintained is to send the entire $\mathcal{W}_{i,j}$ to p_j. This would make $\mathcal{A}_{i,j}$ equal to $\mathcal{K}_i$, and therefore make $\overline{\mathcal{A}_{i,j}}$ equal to $\overline{\mathcal{K}_i}$ and in $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$. Additionally, $\mathcal{W}_{i,j}$ becomes empty. However, this solution is one of the many possible changes to $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$, and not necessarily the optimal one. We leave the method of finding a value for the next message $X_{i,j}$ which should be sent by p_i unspecified at this stage, as it may depend on characteristics of the specific $\mathcal{R}_{\mathcal{F}}$.

The other possible case is that $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right) = T$. Since T is always the last region of $\mathcal{R}_{\mathcal{F}}$, this means $\overline{\mathcal{K}_i}$ is outside any other region $R \in \mathcal{R}_{\mathcal{F}}$. Since T is not necessarily convex, the only option which will guarantee eventual correctness in this case is if p_i sends the entire withheld knowledge to every neighbor it has.

Lastly, we need to address the possibility that although $|\mathcal{W}_{i,j}| = 0$ we will have $\overline{\mathcal{A}_{i,j}}$ which is different from $\overline{\mathcal{K}_i}$. This can happen, *e.g.*, when the withheld knowledge is sent in its entirety and subsequently the local data changes. Notice this possibility results only from our choice to use sufficient statistics rather than sets of vectors: Had we used sets of vectors, $\mathcal{W}_{i,j}$ would not have been empty, and would fall into one of the two cases above. As it stands, we interpret the case of non-empty $\mathcal{W}_{i,j}$ with zero $|\mathcal{W}_{i,j}|$ as if $\overline{\mathcal{W}_{i,j}}$ is in T.

It should be stressed here that if the conditions of Lemma 3.3 hold the peer does not need to do anything even if its knowledge changes. The peer can rely on the correctness of the general results from the previous section which assure that if $\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$ is not the correct answer then eventually one of its neighbors will send it new data and change $\overline{\mathcal{K}_i}$. If, one the other hand, one of the aforementioned cases do occur, then p_i sends a message. This is performed by the **SendMessage** method. If $\overline{\mathcal{K}_i}$ is in T then p_i simply sends all of the withheld data. Otherwise, a message is computed which will assure $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$ are in $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$.

One last mechanism employed in the algorithm is a "leaky bucket" mechanism. This mechanism makes certain no two messages are sent in a period shorter than a constant L. Leaky bucket is often used in asynchronous, event-based systems to prevent event inflation. Every time a message needs to be sent, the algorithm checks how long has it been since the last one was sent. If that time is less than L, the algorithm sets a timer for the reminder of the period and calls **OnChange** again when the timer expires. Note that this mechanism does not enforce any kind of synchronization on the system. It also does not affect correctness: at most it can delay convergence because information would propagate more slowly.

Algorithm 1 Generic Local Algorithm

Input of peer p_i**:** $\mathcal{F}$, $\mathcal{R}_{\mathcal{F}} = \{R_1, R_2, \ldots, T\}$, L, $X_{i,i}$, and N_i
Ad hoc output of peer p_i**:** $\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$
Data structure for p_i**:** For each $p_j \in N_i$ $\overline{X_{i,j}}$, $|X_{i,j}|$, $\overline{X_{j,i}}$, $|X_{i,j}|$, $last\_message$
Initialization: $last\_message \leftarrow -\infty$
On receiving a message $\overline{X}, |X|$ **from** p_j**:**
– $\overline{X_{j,i}} \leftarrow \overline{X}$, $|X_{j,i}| \leftarrow |X|$
On change in $X_{i,i}$, N_i, $\overline{\mathcal{K}_i}$ or $|\mathcal{K}_i|$: call OnChange()
OnChange()
For each $p_j \in N_i$:
– If one of the following conditions occur:
– 1. $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right) = T$ and either $\overline{\mathcal{A}_{i,j}} \neq \overline{\mathcal{K}_i}$ or $|\mathcal{A}_{i,j}| \neq |\mathcal{K}_i|$
– 2. $|\mathcal{W}_{i,j}| = 0$ and $\overline{\mathcal{A}_{i,j}} \neq \overline{\mathcal{K}_i}$
– 3. $\mathcal{A}_{i,j} \notin \mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$ or $\mathcal{W}_{i,j} \notin \mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$
– then
– – call SendMessage(p_j)
SendMessage(p_j)**:**
If $time\,() - last\_message \geq L$
– If $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right) = T$ then the new $\overline{X_{i,j}}$ and $|X_{i,j}|$ are $\overline{\mathcal{W}_{i,j}}$ and $|\mathcal{W}_{i,j}|$, respectively
– Otherwise compute new $\overline{X_{i,j}}$ and $|X_{i,j}|$ such that $\overline{\mathcal{A}_{i,j}} \in \mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$ and either $\overline{\mathcal{W}_{i,j}} \in \mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$ or $|\mathcal{W}_{i,j}| = 0$
– $last\_message \leftarrow time\,()$
– Send $\overline{X_{i,j}}, |X_{i,j}|$ to p_j
Else
– Wait $L - (time\,() - last\_message)$ time units and then call OnChange()

B. Eventual correctness

Proving eventual correctness requires showing that if both the underlying communication graph and the data at every peer cease to change then after some length of time every peer would output the correct result $\mathcal{F}\left(\overline{\mathcal{G}}\right)$; and that this would happen for *any* static communication tree $G(V, E)$, *any* static data $X_{i,i}$ at the peers, and any possible state of the peers.

Proof: [**Eventual Correctness**] Regardless of the state of $\mathcal{K}_i$, $\mathcal{A}_{i,j}$, $\mathcal{W}_{i,j}$, the algorithm will continue to send messages, and accumulate more and more of $\mathcal{G}$ in each $\mathcal{K}_i$ until one of two things happens: One is that for every peer $\mathcal{K}_i = \mathcal{G}$ and thus $\mathcal{A}_{i,j} = \mathcal{K}_i$ for all $p_j \in N_i$. Alternatively, for every p_i $\overline{\mathcal{A}_{i,j}}$ is in $\mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$, which is different than T, and $\mathcal{W}_{i,j}$ is either in $\mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$ as well or is empty. In the former case, $\overline{\mathcal{K}_i} = \overline{\mathcal{G}}$, so every peer obviously computes $\mathcal{F}\left(\overline{\mathcal{K}_i}\right) = \mathcal{F}\left(\overline{\mathcal{G}}\right)$. In the latter case, Theorem 3.1 dictates that $\overline{\mathcal{G}} \in R_\ell$, so $\mathcal{F}\left(\overline{\mathcal{K}_i}\right) = \mathcal{F}\left(\overline{\mathcal{G}}\right)$ too. Finally, provided that every message sent in the algorithm carries the information of at least one input vector to a peer that still does not have it, the number of messages sent between the time the data stops changing and the time in which every peer has the data of all other peers is bounded by $O\left(|V|^2\right)$. ■

C. Local L2 Norm Thresholding

Following the description of a generic algorithm, specific algorithms can be implemented for various functions $\mathcal{F}$. One of the most interesting functions (also dealt with in our previous paper [4]) is that of thresholding the L2 norm of the average vector, i.e., deciding if $\left\|\overline{\mathcal{G}}\right\| \leq \epsilon$.

To produce a specific algorithm from the generic one, the following two steps need to be taken:

1) A respective cover $\mathcal{R}_\mathcal{F}$, needs to be found
2) A method for finding $\overline{X_{i,j}}$ and $|X_{i,j}|$ which assures that both $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$ are in R needs to be formulated

In the case of L2 thresholding, the area for which $\mathcal{F}$ outputs $true$ – the inside of an ϵ circle – is convex. This area is denoted R_{in}. The area outside the ϵ-circle can be divided by randomly selecting unit vectors $\hat{u_1}, \ldots, \hat{u_\ell}$ and then drawing the half-spaces $H_j = \{\vec{x} : \vec{x} \cdot \hat{u_j} \geq \epsilon\}$. Each half-space is convex. Also, they are entirely outside the ϵ circle, so $\mathcal{F}$ is constant on every H_j. $\{R_{in}, H_1, \ldots, H_\ell, T\}$ is, thus, a respective cover. Furthermore, by increasing ℓ, the area between the halfspaces and the circle or the tie area can be minimized to any desired degree.

It is left to describe how the **SendMessage** method computes a message that forces $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$ into the region which contains $\overline{\mathcal{K}_i}$ if they are not in it. A related algorithm, Majority-Rule [1], suggests sending all of the withheld knowledge in any case. However, experiments with dynamic data hint this method may be unfavorable. If all or most of the knowledge is sent and the data later changes the withheld knowledge becomes the difference between the old and the new data. This difference tends to be far more noisy than the original data. Thus, while the algorithm makes certain $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$ are brought into the same region as $\overline{\mathcal{K}_i}$, it still makes an effort to maintain some withheld knowledge.

Although it may be possible to optimize the size of $|\mathcal{W}_{i,j}|$ we take the simple and effective approach of testing an exponentially decreasing sequence of $|\mathcal{W}_{i,j}|$ values, and then choosing the first such value satisfying the requirements for $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$. When a peer p_i needs to send a message, it first sets the new $\overline{X_{i,j}}$ to $\frac{|\mathcal{K}_i|\overline{\mathcal{K}_i} - |X_{j,i}|\overline{X_{j,i}}}{|\mathcal{K}_i| - |X_{j,i}|}$. Then, it tests a sequence of values for $|X_{i,j}|$. Clearly, $|X_{i,j}| = |\mathcal{K}_i| - |X_{j,i}|$ translates to an empty withheld knowledge and must concur with the conditions of Lemma 3.3. However, the algorithm begins with $|X_{i,j}| = \frac{|\mathcal{K}_i| - |X_{j,i}|}{2}$ and only gradually increases the weight, trying to satisfy the conditions without sending all data.

Algorithm 2 Local L2 Thresholding

Input of peer p_i**:** ϵ, L, $X_{i,i}$, N_i, ℓ
Global constants: A random seed s
Data structure for p_i**:** For each $p_j \in N_i$ $\overline{X_{i,j}}$, $|X_{i,j}|$, $\overline{X_{j,i}}$, $|X_{i,j}|$, $last\_message$
Output of peer p_i**:** 0 if $\left\|\overline{\mathcal{K}_i}\right\| \leq \epsilon$, 1 otherwise
Computation of $\mathcal{R}_\mathcal{F}$**:**
Let $R_{in} = \{\vec{x} : \|\vec{x}\| \leq \epsilon\}$
Let $\hat{u_1}, \ldots, \hat{u_\ell}$ be pseudo-random unit vectors and let $H_j = \{\vec{x} : \vec{x} \cdot \hat{u_j} \geq \epsilon\}$
$\mathcal{R}_\mathcal{F} = \{R_{in}, H_1, \ldots, H_\ell, T\}$.
Computation of $|X_{i,j}|$ **and** $\overline{X_{i,j}}$**:**
$\overline{X_{i,j}} \leftarrow \frac{|\mathcal{K}_i|\overline{\mathcal{K}_i} - |X_{j,i}|\overline{X_{j,i}}}{|\mathcal{K}_i| - |X_{j,i}|}$
$w \leftarrow |X| \leftarrow |\mathcal{K}_i| - |X_{j,i}|$
Do
– $w \leftarrow \lfloor \frac{w}{2} \rfloor$
– $|X_{i,j}| \leftarrow |\mathcal{K}_i| - |X_{j,i}| - w$
While ($\overline{\mathcal{A}_{i,j}} \notin \mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$ or $\overline{\mathcal{W}_{i,j}} \notin \mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$ and $|\mathcal{W}_{i,j}| \neq 0$)
Initialization: $last\_message \leftarrow -\infty$, compute $\mathcal{R}_\mathcal{F}$
On receiving a message $\overline{X}, |X|$ **from** p_j**:**
– $\overline{X_{j,i}} \leftarrow \overline{X}$, $|X_{j,i}| \leftarrow |X|$
On change in $X_{i,i}$, N_i, $\overline{\mathcal{K}_i}$ or $|\mathcal{K}_i|$: call OnChange()
OnChange()
For each $p_j \in N_i$:
– If one of the following conditions occur:
– 1. $\mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right) = T$ and either $\overline{\mathcal{A}_{i,j}} \neq \overline{\mathcal{K}_i}$ or $|\mathcal{A}_{i,j}| \neq |\mathcal{K}_i|$
– 2. $|\mathcal{W}_{i,j}| = 0$ and $\overline{\mathcal{A}_{i,j}} \neq \overline{\mathcal{K}_i}$
– 3. $\mathcal{A}_{i,j} \notin \mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$ or $\mathcal{W}_{i,j} \notin \mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$
– then
– – call SendMessage(p_j)
SendMessage(p_j)**:**
If $time\,() - last\_message \geq L$
– If $\mathcal{R}_\mathcal{F}\left(\overline{\mathcal{K}_i}\right) = T$ then the new $\overline{X_{i,j}}$ and $|X_{i,j}|$ are $\overline{\mathcal{W}_{i,j}}$ and $|\mathcal{W}_{i,j}|$, respectively
– Otherwise compute new $\overline{X_{i,j}}$ and $|X_{i,j}|$
– $last\_message \leftarrow time\,()$
– Send $\overline{X_{i,j}}, |X_{i,j}|$ to p_j
Else
– Wait $L - (time\,() - last\_message)$ time units and then call OnChange()

V. Reactive Algorithms

The previous section described an efficient generic local algorithm, capable of computing any function even when the data and system are constantly changing. In this section, we leverage this powerful tool to create a framework for producing and maintaining various data mining models. This framework is simpler than the current methodology of inventing a specific distributed algorithm for each problem and may be as efficient as its counterparts.

The basic idea of the framework is to employ a simple, costly, and possibly inaccurate *convergecast* algorithm in which a single peer samples data from the network and then computes, based on this "best-effort" sample, a data mining model. Then, this model is *broadcast* to the entire network; again, a technique which might be costly. Once, every peer is informed with the current model, a local algorithm, which is an instantiation of the generic algorithm is used in order to monitor the quality of the model. If the model is not sufficiently accurate or the data has changed to the degree that the model no longer describes it, the monitoring algorithm alerts and triggers another cycle of data collection. It is also possible to tune the algorithm by increasing the sample size if the alerts are frequent and decreasing it when they are infrequent. Since the monitoring algorithm is eventually correct, eventual convergence to a sufficiently accurate model is very likely. Furthermore, when the data only goes through stationary changes, the monitoring algorithm triggers false alerts infrequently and hence can be extremely efficient. Thus, the overall cost of the framework is low.

We describe two instantiations of this basic framework, each highlighting a different aspect. First we discuss the problem of computing the mean input vector, to a desired degree of accuracy. Then, we present an algorithm for computing a variant of the k-means clusters suitable for dynamic data.

A. Mean Monitoring

The problem of monitoring the mean of the input vectors has direct applications to many data analysis tasks. The objective in this problem is to compute a vector $\overline{\mu}$ which is a good approximation for $\overline{\mathcal{G}}$. Formally, we require that $\left\|\overline{\mathcal{G}} - \overline{\mu}\right\| \leq \epsilon$ for a desired value of ϵ.

For any given estimate $\overline{\mu}$, monitoring whether $\left\|\overline{\mathcal{G}} - \overline{\mu}\right\| \leq \epsilon$ is possible via direct application of the L2 thresholding algorithm from Section IV-C. Every peer p_i subtracts $\overline{\mu}$ from every input vector in $X_{i,i}$. Then, the peers jointly execute L2 Norm Thresholding over the modified data. If the resulting average is inside the ϵ-circle then $\overline{\mu}$ is a sufficiently accurate approximation of $\overline{\mathcal{G}}$; otherwise, it is not.

The basic idea of the mean monitoring algorithm is to employ a convergecast-broadcast process in which the convergecast part computes the average of the input vectors and the broadcast part delivers the new average to all the peers. The trick is that, before a peer sends the data it collected up the convergecast tree, it waits for an indication that the current $\overline{\mu}$ is not a good approximation of the current data. Thus, when the current $\overline{\mu}$ is a good approximation, convergecast is slow and only progresses as a result of false alerts. During this time, the cost of the convergecast process is negligible compared to that of the L2 thresholding algorithm. When, on the other hand, the data does change, all peers alert almost immediately. Thus, convergecast progresses very fast, reaches the root, and initiates the broadcast phase. Hence, a new $\overline{\mu}$ is delivered to every peer, which is a more updated estimate of $\overline{\mathcal{G}}$.

The details of the mean monitoring algorithm are given in Algorithm 3. One detail is that of an alert mitigation constant, τ, selected by the user. The idea here is that an alert should persist for a given period of time before the convergecast advances. Experimental evidence suggests that setting τ to even a fraction of the average edge delay greatly reduces the number of convergecasts without incurring a significant delay in the updating of $\overline{\mu}$.

A second detail is the separation of the data used for alerting – the input of the L2 thresholding algorithm – from that which is used for computing the new average. If the two are the same then the new average may be biased. This is because an alert, and consequently an advancement in the convergecast, is bound to be more frequent when the local data is extreme. Thus, the initial data, and later every new data, is randomly associated with one of two buffers: R_i, which is used by the L2 Thresholding algorithm, and T_i, on whom the average is computed when convergecast advances.

A third detail is the implementation of the convergecast process. First, every peer tracks changes in the knowledge of the underlying L2 thresholding algorithm. When it moves from inside the ϵ-circle to outside the ϵ-circle the peer takes note of the time, and sets a timer to τ time units. When a timer expires or when a data message is received from one of its neighbors p_i checks if currently there is an alert and if it was recorded τ or more time units ago. If so, it counts the number of its neighbors from whom it received a data message. If it received data messages from all of its neighbors, the peer moves to the broadcast phase, computes the average of its own data and of the received data and sends it to itself. If it has received data messages from all but one of the neighbors then this one neighbor becomes the peer's parent in the convergecast tree; the peer computes the average of its own and its other neighbors' data, and sends the average with its cumulative weight to the parent. Then, it moves to the broadcast phase. If two or more of its neighbors have not yet sent a data messages p_i keeps waiting.

Lastly, the broadcast phase is fairly straightforward. Every peer which receives the new $\overline{\mu}$ vector, updates its data by subtracting it from every vector in R_i and transfers those vectors to the underlying L2 thresholding algorithm. Then, it re-initializes the buffers for the data messages and sends the new $\overline{\mu}$ vector to its other neighbors and changes the status to convergecast. There could be one situation in which a peer receives a new $\overline{\mu}$ vector even though it is already in the convergecast phase. This happens when two neighbor peers concurrently become roots of the convergecast tree (i.e., when each of them concurrently sends the last convergecast message to the other). To break the tie, a root peer p_i which receives $\overline{\mu}$ from a neighbor p_j while in the convergecast phase ignores the message if $i > j$ it ignores the message. Otherwise if $i < j$ p_i treats the message just as it would in the broadcast phase.

B. k-Means Monitoring

We now turn to a more complex problem, that of computing the k-means of distributed data. The classic formulation of the k-means algorithm is a two step recursive process in which every data point is first associated with the nearest of k centroids, and then every centroid is moved to the average of the points associated with it – until the average is the same

Algorithm 3 Mean Monitoring

Input of peer p_i: ϵ, L, $X_{i,i}$, the set of neighbors N_i, an initial vector $\overline{\mu_0}$, an alert mitigation constant τ.
Output available to every peer p_i: An approximated means vector $\overline{\mu}$
Data structure of peer p_i: Two sets of vectors R_i and T_i, a timestamp $last\_change$, flags: $alert$, $root$, and $phase$, for each $p_j \in N_i$, a vector $\overline{v}_j$ and a counter c_j
Initialization:
Set $\overline{\mu} \leftarrow \overline{\mu_0}$, $alert \leftarrow false$, $phase \leftarrow convergecast$
Split $X_{i,i}$ evenly between R_i and T_i
Initialize an L2 thresholding algorithm with the input ϵ, L, $\{\overline{x} - \overline{\mu} : \overline{x} \in R_i\}$, N_i
Set $\overline{v_i}, c_i$ to $\overline{T_i}, |T_i|$, respectively, and $\overline{v_j}, c_j$ to $\overline{0}, 0$ for all other $p_j \in N_i$
On addition of a new vector $\overline{x}$ to $X_{i,i}$:
Randomly add $\overline{x}$ to either R_i or T_i
If $\overline{x}$ was added to R_i, update the input of the L2 thresholding algorithm to $\{\overline{x} - \overline{\mu} : \overline{x} \in R_i\}$
Otherwise, update v_i and c_i.
On change in $\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$ of the L2 thresholding algorithm:
If $\left\|\overline{\mathcal{K}_i}\right\| \geq \epsilon$ and $alert = false$ then
– set $last\_change \leftarrow time()$
– set $alert \leftarrow true$
– set a timer to τ time units
If $\left\|\overline{\mathcal{K}_i}\right\| < \epsilon$ then
– Set $alert \leftarrow false$
On receiving a data message $\overline{v}, c$ from $p_j \in N_i$:
Set $\overline{v_j} \leftarrow \overline{v}$, $c_j \leftarrow c$
Call Convergecast
On timer expiry or call to Convergecast:
If $alert = false$ return
If $time() - last\_change < \tau$ set timer to $time() + \tau - last\_change$ and return
If for all $p_k \in N_i$ except for one $c_k \neq 0$
– Let $s = \sum_{p_j \in N_i} c_j$, $\overline{s} = \sum_{p_j \in N_i} \frac{c_j}{s} \overline{v_j}$
– Send $s, \overline{s}$ to p_l
– Set $phase \leftarrow Broadcast$
If for all $p_k \in N_i$ $c_k \neq 0$
– Let $s = \sum_{p_j \in N_i} c_j$, $\overline{s} = \sum_{p_j \in N_i} \frac{c_j}{s} \overline{v_j}$
– Set $phase \leftarrow Convergecast$
– Send $\overline{\mu}$ to all $p_k \in N_i$
On receiving $\overline{\mu'}$ from $p_j \in N_i$:
If $phase = convergecast$ and $i > j$ then return
Set $\overline{\mu} \leftarrow \overline{\mu'}$
Replace the input of the L2 thresholding algorithm with $\{\overline{x} - \overline{\mu} : \overline{x} \in R_i\}$
Set $phase \leftarrow convergecast$ and set all c_j to 0
Send $\overline{\mu}$ to all $p_k \neq p_j \in N_i$
Other than that follow the L2 thresholding algorithm

as the centroid. To make the algorithm suitable for a dynamic data setup, we relax the stopping criteria. In our formulation, a solution is considered admissible when the average of point is within an ϵ-distance of the centroid with whom they are associated.

Similar to the mean monitoring, the k-means monitoring algorithm (Algorithm. 4) is performed in a cycle of convergecast and broadcast. The algorithm, however, is different in some important respects. First, instead of taking part of just one execution of L2 thresholding, each peer takes part in k such executions – one per centroid. The input of the ℓ^{th} execution are those points in the local data set $X_{i,i}$ for which the ℓ^{th} centroid, $\overline{c_\ell}$, is the closest. Thus, each execution monitors whether one of the centroids needs to be updated. If even one execution discovers that the norm of the respective knowledge $\left\|\overline{\mathcal{K}_i^\ell}\right\|$ is greater than ϵ, the peer alerts, and if the alert persists for τ time units the peer advances the convergecast process.

Another difference between k-means monitoring and mean monitoring is the statistics collected during convergecast. In k-means monitoring, that statistics is a sample of size b (dictated by the user) from the data. Each peer samples with returns from the samples it received from its neighbors, and from its own data, such that the probability of sampling a point is proportional to a weight. The result of this procedure is that every input point stands an equal chance to be included in the sample that arrives to the root. The root then computes the k-means on the sample, and sends the new centroids in a broadcast message.

VI. Experimental Validation

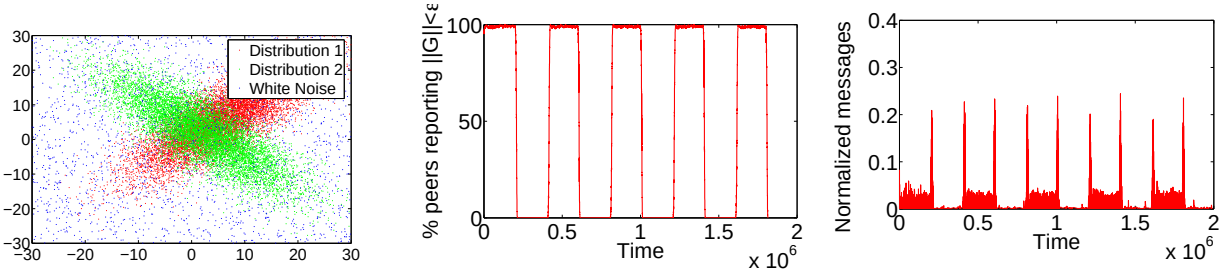
To validate the performance of our algorithms we conducted experiments on a simulated network of thousands of peers. In this section we discuss the experimental setup and analyze the performance of the algorithms.

A. Experimental Setup

Our implementation makes use of the Distributed Data Mining Toolkit (DDMT)[1]– a distributed data mining development environment from DIADIC research lab at UMBC. DDMT uses topological information which can be generate by BRITE[2], a universal topology generator from Boston University. In our simulations we used topologies generated according to the *Barabasi Albert (BA)* model, which is often considered a reasonable model for the Internet. BA also defines delays for network edges, which are the basis for our time measurement[3]. On top of the network generated by BRITE, we overlayed a spanning tree.

The data used in the simulations was generated using a mixture of Gaussians in $\mathbb{R}^d$. Every time a simulated peer needed an additional data point, it sampled d Gaussians and multiplied the resulting vector with a $d \times d$ covariance matrix in which the diagonal elements were all 1.0's while the off-diagonal elements were chosen uniformly between 1.0 and

[1]http://www.umbc.edu/ddm/wiki/software/DDMT
[2]http://www.cs.bu.edu/brite/
[3]Wall time is meaningless when simulating thousands of computers on a single PC.



(a) Typical data set

(b) Typical changes in the percent of peers with $\left\|\overline{\mathcal{K}_i}\right\| \leq \epsilon$

(c) Typical messaging throughout an experiment

Fig. 2. A typical experiment is run for 10 equal length epochs. The epochs have very similar means, and very large variance. Quality and overall cost are measured across the entire experiment – including transitional phases.

2.0. Alternatively, 10% of the points were chosen uniformly at random in the range of $\mu \pm 3\sigma$. At controlled intervals, the means of the Gaussians were changed, thereby creating an epoch change. A typical data in two dimensions can be seen in Figure 2(a). We preferred synthetic data because of the large number of factors (twelve, in our analysis) which influence the behavior of an algorithm, and the desire to perform a tightly controlled experiment in order to understand the behavior of a complex algorithm which operates in an equally as complex environment.

The two most important qualities measured in our experiments are the *quality* of the result and the *cost* of the algorithm. Quality is defined differently for the L2 thresholding algorithm, the mean monitoring algorithm, and the k-means algorithm.

For the L2 thresholding algorithm, quality is measured in terms of the number of peers correctly computing an alert *i.e.* the percentage of peers for whom $\left\|\overline{\mathcal{K}_i}\right\| < \epsilon$ when $\left\|\overline{\mathcal{G}}\right\| < \epsilon$, and the percentage of peers for whom $\left\|\overline{\mathcal{K}_i}\right\| \geq \epsilon$ when $\left\|\overline{\mathcal{G}}\right\| \geq \epsilon$. We measure the maximal, average and minimal quality over all the peers (averaged over a number of different experiments). Quality is reported in three different scenarios: overall quality, averaged over the entire experiment; and quality on stationary data, measured separately for periods in which the mean of the data is inside the ϵ-circle $\left(\left\|\overline{\mathcal{G}}\right\| < \epsilon\right)$ and for periods in which the means of the data is outside the circle $\left(\left\|\overline{\mathcal{G}}\right\| \geq \epsilon\right)$.

For the mean monitoring algorithm, quality is the average distance between $\overline{\mathcal{G}}$ and the computed mean vector $\overline{\mu}$. We plot, separately, the overall quality (during the entire experiment) and the quality after the broadcast phase ended.

Lastly, for the k-means algorithm, quality is defined as the distance between the solution of our algorithm and that computed by a centralized algorithm, given all the data of all of the peers.

We have measured the cost of the algorithm according to the frequency in which messages are sent by each peer. Because of the leaky bucket mechanism which is part of the algorithm, the rate of messages per average peer is bounded by two for every L time units (one to each neighbor, for an average of two neighbors per peer). The trivial algorithm that floods every change in the data would send messages at this rate. The communication cost of our algorithms is thus defined in terms of normalized messages - the portion of this maximal rate which the algorithm uses. Thus, 0.1 normalized messages means that nine times out of ten the algorithm manages to avoid sending a message. We report both overall cost, which includes the stationary and transitional phases of the experiment (and thus is necessarily higher), and the monitoring cost, which only refers to stationary periods. The monitoring cost is the cost paid by the algorithm even if the data remains stationary; hence, it measures the "wasted effort" of the algorithm. We also separate, where appropriate, messages pertaining to the computation of the L2 thresholding algorithm from those used for convergecast and broadcast of statistics.

There are many factors which may influence the performance of the algorithms. First, are those pertaining to the data: the number of dimensions d, the covariance σ, and the distance between the means of the Gaussians of the different epochs (the algorithm is oblivious to the actual values of the means), and the length of the epochs T. Second, there are factors pertaining to the system: the topology, the number of peers, and the size of the local data. Last, there are control arguments of the algorithm: most importantly ϵ – the desired alert threshold, and then also L – the maximal frequency of messages. In all the experiments that we report in this section, one parameter of the system was changed and the others were kept at their default values. The default values were : number of peers = 1000, $|X_{i,i}| = 800$, $\epsilon = 2$, $d = 5$, $L = 500$ (where the average edge delay is about 1100 time units), and the Frobenius norm of the covariance of the data $\|\sigma\|_F$ at 5.0. We selected the distance between the means so that the rates of false negatives and false positives are about equal. More specifically, the means for one of the epochs was +2 along each dimension and for the other it was -2 along each dimension. For each selection of the parameters, we ran the experiment for a long period of simulated time, allowing 10 epochs to occur.

A typical experiment is described in Figure 2(b) and 2(c). In the experiment, after every 2×10^5 simulator ticks, the data distribution is changed, thereby creating an epoch change. To start with, every peer is given the same mean as the mean of the Gaussian. Thus a very high percentage ($\sim$ 100 %) of the peers states that $\left\|\overline{\mathcal{G}}\right\| < \epsilon$. After the aforesaid number (2×10^5) of simulator ticks, we change the Gaussian without changing

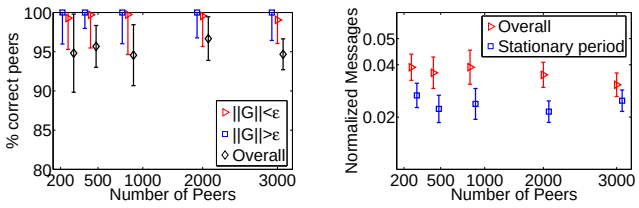
Algorithm 4 k-Means Monitoring

Input of peer p_i**:** ϵ, L, $X_{i,i}$, the set of immediate neighbors N_i, an initial guess for the centroids C_0, a mitigation constant τ, the sample size b.
Output of peer p_i**:** k centroids such that the average of the points assigned to every centroid is within ϵ of that centroid.
Data structure of peer p_i**:** A partitioning of $X_{i,i}$ into k sets $X_{i,i}^1 \dots X_{i,i}^k$, a set of centroids $C = \{\overline{c_1}, \dots, \overline{c_k}\}$, for each centroid $j = 1, \dots, k$, a flag $alert_j$, a times tamp $last\_change_j$, a buffer B_j and a counter b_j, a flag $root$ and a flag $phase$.
Initialization:
Set $C \leftarrow C_0$. Let $X_{i,i}^j = \left\{ \overline{x} \in X_{i,i} : \overline{c_j} = \arg\min_{c \in C} \|\overline{x} - \overline{c}\| \right\}$. Initialize k instances of the L2 thresholding algorithm, such that the j^{th} instance has input ϵ, α, L, $\left\{ \overline{x} - \overline{c_j} : \overline{x} \in X_{i,i}^j \right\}$, N_i. For all $p_j \in N_i$, set $b_j \leftarrow 0$, for all $j = 1, \dots, k$ set $alert_j \leftarrow false$, $last\_change_j \leftarrow -\infty$, and $phase \leftarrow convergecast$
On addition of a new vector $\overline{x}$ **to** $X_{i,i}$**:**
Find the c_j closest to $\overline{x}$ and add $\overline{x} - \overline{c_j}$ to the j^{th} L2 thresholding instance.
On removal of a vector $\overline{x}$ **from** $X_{i,i}$**:**
Find the c_j closest to $\overline{x}$ and remove $\overline{x} - \overline{c_j}$ from the j^{th} L2 thresholding instance.
On change in $\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$ **of the** j^th **instance of the L2 thresholding algorithm:**
If $\left\|\overline{\mathcal{K}_i}\right\| \geq \epsilon$ and $alert_j = false$ then set $last\_change_j \leftarrow time()$, $alert_j \leftarrow true$, and set a timer to τ time units
If $\left\|\overline{\mathcal{K}_i}\right\| < \epsilon$ then set $alert_j \leftarrow false$
On receiving B, b **from** $p_j \in N_i$**:**
Set $B_j \leftarrow B$, $b_j \leftarrow b$ and call Convergecast
On timer expiry or call to Convergecast:
If for all $\ell \in [1, \dots, k]$ $alert_\ell = false$ then return
Let $t \leftarrow Min_{\ell=1\dots k} \{last\_message_\ell : alert_\ell = true\}$
Let A be a set of b samples returned by **Sample**
If $time() < t + \tau$ then set a timer to $t + \tau - time()$ and return
If for all $p_k \in N_i$ except for one $b_k \neq 0$
– Set $root \leftarrow false$, $phase \leftarrow Broadcast$
– Send $A, |X_{i,i}| + \sum_{m=1\dots} b_m$ to p_ℓ and return
If for all $p_k \in N_i$ $b_k \neq 0$
– Let C' be the centroids resulting from computing the k-means clustering of A
– Set $root \leftarrow true$
– Send C' to self and return
On receiving C' **from** $p_j \in N_i$ **or from self:**
If $phase = convergecast$ and $i > j$ then return
Set $C \leftarrow C'$
For $j = 1 \dots k$ set $X_{i,i}^j = \left\{ \overline{x} \in X_{i,i} : \overline{c_j} = \arg\min_{c \in C} \|\overline{x} - \overline{c}\| \right\}$
For $j = 1 \dots |N_i|$ set $b_j \leftarrow 0$
Send C to all $p_k \neq p_j \in N_i$
Set $phase \leftarrow Convergecast$
On call to Sample:
Return a random sample from $X_{i,i}$ with probability $1/\left(1 + \sum_{m=1\dots|N_i|} b_m\right)$ or from a buffer B_j with probability $b_j/\left(|X_{i,i}| + \sum_{m=1\dots|N_i|} b_m\right)$

the mean given to each peer. Thus, for the next epoch, we see that a very low percentage of the peers ($\sim$ 0 %) output that $\left\|\overline{\mathcal{G}}\right\| < \epsilon$. For the cost of the algorithm in Figure 2(c), we see that messages exchanged during the stationary phase is low. Many messages are, however, exchanged as soon as the epoch changes. This is expected since all the peers need to communicate in order to get convinced that the distribution has indeed changed. The number of messages decreases once the distribution becomes stable again.

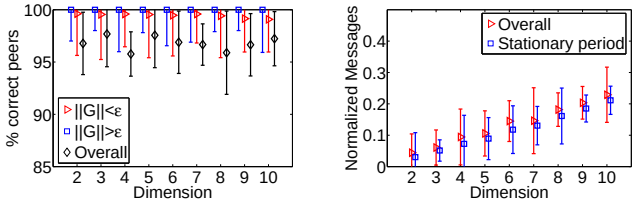
B. Experiments with Local L2 Thresholding Algorithm

The L2 thresholding algorithm is the simplest one we present here. In our experiments, we use the L2 thresholding to establish the scalability of the algorithms with respect to both the number of peers and the dimensionality of the data, and the dependency of the algorithm on the main parameters – the norm of the covariance σ, the size of the local data set, the tolerance ϵ, and the bucket size L.



(a) Quality vs. number of peers (b) Cost vs. number of peers

Fig. 3. Scalability of Local L2 algorithm with respect to the number of peers.



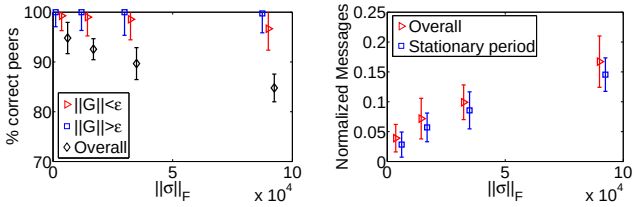
(a) Quality vs. dimension (b) Cost vs. dimension

Fig. 4. Scalability of Local L2 algorithm with respect to the dimension of the domain.

In Figures 3 and 4, we analyze the scalability of the local L2 algorithm. As Figure 3(a) and Figure 3(b) show, the average quality and cost of the algorithm converge to a constant as the number of peers increase. This typifies local algorithms – because the computation is local, the total number of peers do not affect performance. Hence, there could be no deterioration in quality or cost. Similarly, the number of messages per peer become a constant – typical to local algorithms. Figure 4(a) and Figure 4(b) show the scalability with respect to the dimension of the problem. As shown in the figures, quality does not deteriorate when the dimension of the problem is increased. Also note that the cost increases approximately linearly with the dimension. This independence of the quality can be explained if one thinks of what the algorithm does in terms of domain linearization. We hypothesis that when

the mean of the data is outside the circle, most peers tend to select the same half-space. If this is true then the problem is projected along the vector defining that half-space – i.e., becomes uni-dimensional. Inside the circle, the problem is again uni-dimensional: If thought about in terms of the polar coordinate system (rooted at the center of the circle), then the only dimension on which the algorithm depends is the radius. The dependency of the cost on the dimension stems from the linear dependence of the variance of the data on the number of Gaussians, the variance of whom is constant. This was proved in experiments not included here.

In Figures 5, 6, 7 and 8 we explore the dependency of the L2 algorithm on different parameters *viz.* Frobenius norm of the covariance of the data σ ($\|\sigma\|_F = \sum_{i=1\ldots m}\sum_{j=1\ldots n}|\sigma_{i,j}|^2$), the size of the local data buffer $|X_{i,i}|$, the alert threshold ϵ, and the size of the leaky bucket L. As noted earlier, in each experiment one parameter was varied and the rest were kept at their default values.

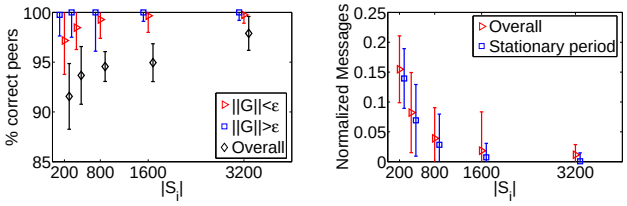


(a) Quality vs. $\|\sigma\|_F$ (b) Cost vs. $\|\sigma\|_F$

Fig. 5. Dependency of cost and quality of L2 thresholding on $\|\sigma\|_F$. Quality is defined by the percentage of peers correctly computing an alert (separated for epochs with $\left\|\overline{\mathcal{G}}\right\|$ less and more than ϵ). Cost is defined as the portion of the leaky buckets intervals that are used. Both overall cost and cost of just the stationary periods are reported. Overall measurements include the transitional period too.

The first pair of figures, Figure 5(a) and Figure 5(b), outline the dependency of the quality and the cost on the covariance of the data ($\sigma = A\overline{E}$) where A is the covariance matrix and $\overline{E}$ is the variance of the gaussians. Matrix A is as defined in Section VI-A while $\overline{E}$ is the column vector representing the variance of the gaussians and takes the values 5, 10, 15 or 25. For epochs with $\left\|\overline{\mathcal{G}}\right\| < \epsilon$, the maximal, the average, and the minimal quality in every experiment decrease linearly with the variance (from around 99% on average to around 96%). Epochs with $\left\|\overline{\mathcal{G}}\right\| > \epsilon$, on the other hand, retained very high quality, regardless of the level of variance. The overall quality also decreases linearly from around 97% to 84%, apparently resulting from slower convergence on every epoch change. As for the cost of the algorithm, this increases as the square root of $\|\sigma\|_F$ (i.e., linear to the variance), both for the stationary and overall period. Nevertheless, even with the highest variance, the cost stayed far from the theoretical maximum of two messages per peer per leaky bucket period.

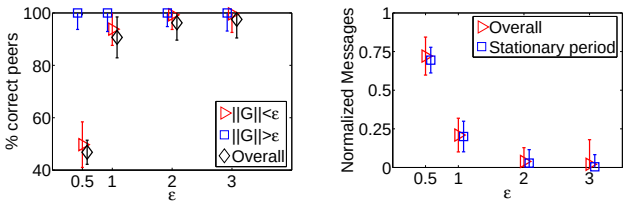
The second pair of figures, Figure 6(a) and Figure 6(b), shows that the variance can be controlled by increasing the local data. As $|X_{i,i}|$ increases, the quality increases, and cost decreases, proportional to $\sqrt{|X_{i,i}|}$. The cause of that is clearly the relation of the variance of an i.i.d. sample to the sample size which is inverse of the square root.



(a) Quality vs. $|X_{i,i}|$ (b) Cost vs. $|X_{i,i}|$

Fig. 6. Dependency of cost and quality of L2 thresholding on $|X_{i,i}|$. Quality is defined by the percentage of peers correctly computing an alert (separated for epochs with $\left\|\overline{\mathcal{G}}\right\|$ less and more than ϵ). Cost is defined as the portion of the leaky buckets intervals that are used. Both overall cost and cost of just the stationary periods are reported. Overall measurements include the transitional period too.

The third pair of figures, Figure 7(a) and Figure 7(b), present the effect of changing ϵ on both the cost and quality of the algorithm. As can be seen, below a certain point, the number of false positives grows drastically. The number of false negatives, on the other hand, remains constant regardless of ϵ. When ϵ is about two, the distances of the two means of the data (for the two epochs) from the boundary of the circle are approximately the same and hence the rates of false positives and false negatives are approximately the same too. As ϵ decreases, it becomes increasingly difficult to judge if the mean of the data is inside the smaller circle and increasingly easier to judge that the mean is outside the circle. Thus, the number of false positives increase. The cost of the algorithm decreases linearly as ϵ grows from 0.5 to 2.0, and reaches nearly zero for $\epsilon = 3$. Note that even for a fairly low $\epsilon = 0.5$, the number of messages per peer per leaky bucket period is around 0.75, which is far less than the theoretical maximum of 2.

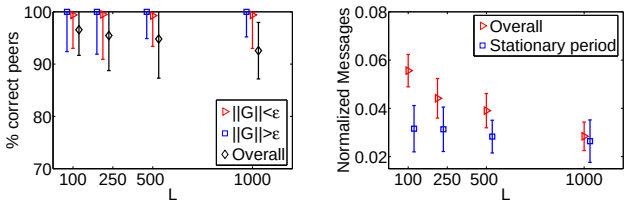


(a) Quality vs. ϵ (b) Cost vs. ϵ

Fig. 7. Dependency of cost and quality of L2 thresholding on ϵ. Quality is defined by the percentage of peers correctly computing an alert (separated for epochs with $\left\|\overline{\mathcal{G}}\right\|$ less and more than ϵ). Cost is defined as the portion of the leaky buckets intervals that are used. Both overall cost and cost of just the stationary periods are reported. Overall measurements include the transitional period too.

Figure 8(a) and Figure 8(b) explore the dependency of the quality and the cost on the size of the leaky bucket L. Interestingly, the reduction in cost here is far faster than the reduction in quality, with the optimal point (assuming 1:1 relation between cost and quality) somewhere between 100 time units and 500 time units. It should be noted that the average delay BRITE assigned to an edge is around 1100 time units. This shows that even a very permissive leaky bucket mechanism is sufficient to greatly limit the number

of messages.



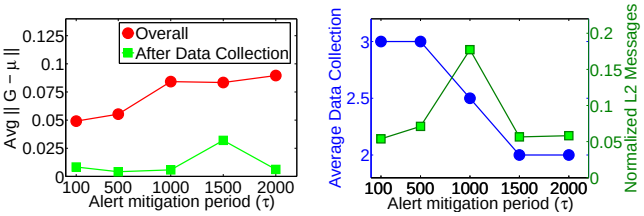
(a) Quality vs. L (b) Cost vs. L

Fig. 8. Dependency of cost and quality of L2 thresholding on L. Quality is defined by the percentage of peers correctly computing an alert (separated for epochs with $\left\|\overline{\mathcal{G}}\right\|$ less and more than ϵ). Cost is defined as the portion of the leaky buckets intervals that are used. Both overall cost and cost of just the stationary periods are reported. Overall measurements include the transitional period too.

We conclude that the L2 thresholding provides a moderate rate of false positives even for noisy data and an excellent rate of false negatives regardless of the noise. It requires little communication overhead during stationary periods. Furthermore, the algorithm is highly scalable – both with respect to the number of peers and dimensionality – because performance is independent of the number of peers and dimension of the problem.

C. Experiments with Means-Monitoring

Having explored the effects of the different parameters of the L2 thresholding algorithm, we now shift our focus on the experiments with the mean monitoring algorithm. We have explored the three most important parameters that affect the behavior of the mean monitoring algorithm: τ – the alert mitigation period, T – the length of an epoch, and ϵ – the alert threshold.

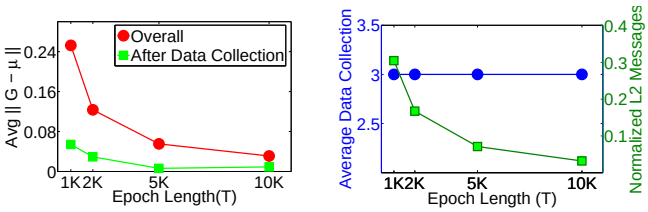


(a) Quality vs. τ (b) Cost vs. τ

Fig. 9. Dependency of cost and quality of mean monitoring on the alert mitigation period τ.

Figure 9, 10 and 11 summarize the results of these experiments. As can be seen, the quality, measured by the distance of the actual means vector $\overline{\mathcal{G}}$ from the computed one $\overline{\mu}$ is excellent in all three graphs. Also shown are the cost graphs with separate plots for the L2 messages (on the right axis) and the number of convergecast rounds – each costs two messages per peer on average – (on the left axis) per epoch.

In Figure 9(a), the average distance between $\overline{\mathcal{G}}$ and $\overline{\mu}$ decreases as the alert mitigation period (τ) is decreased for the entire length of the experiment. This is as expected, since, with a smaller τ, the peers can rebuild the model more frequently, resulting in more accurate models. On the other hand, the quality after the data collection is extremely good and is independent of τ. With increasing τ, the number of convergecast rounds per epoch decreases (from three to two on average) as shown in Figure 9(b). In our analysis, this results from a decrease in the number of false alerts.

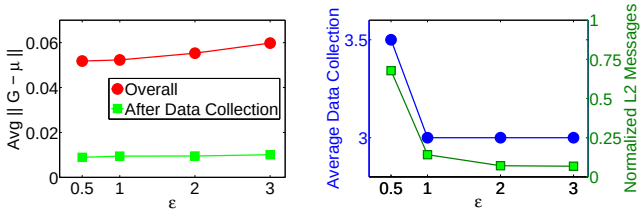


(a) Quality vs. epoch length (b) Cost vs. epoch length

Fig. 10. Dependency of cost and quality of mean monitoring on the length of epoch T.

Figure 10(a) depicts the relation of the quality (both overall and stationary periods) to T. The average distance between the estimated mean vector and the actual one decreases as the epoch length T increases. The reason is the following: at each epoch, several convergecast rounds usually occur. The later the round is, the less polluted is the data by remnants of the previous epoch – and thus the more accurate is $\overline{\mu}$. Thus, when the epoch length increases, the proportion of these later $\overline{\mu}$'s, which are highly accurate, increases in the overall quality leading to a more accurate average. Figure 10(b) shows a similar trend for the cost incurred. One can see that the number of L2 messages decrease as T increases. Clearly, the more accurate $\overline{\mu}$ is, the less monitoring messages are sent. Therefore with increasing T, the quality increases and cost decreases in the later rounds and these effects are reflected in the figures.

Finally, the average distance between $\overline{\mathcal{G}}$ and $\overline{\mu}$ decreases as ϵ decreases. This is as expected, since with decreasing ϵ, the L2 algorithm ensures that these two quantities be brought closer to each other and thus the average distance between them decreases. The cost of the algorithm, however, shows the reverse trend. This result is intuitive – with increasing ϵ, the algorithm has a larger region in which to bound the global average and thus the problem becomes easier, and hence less costly, to solve.



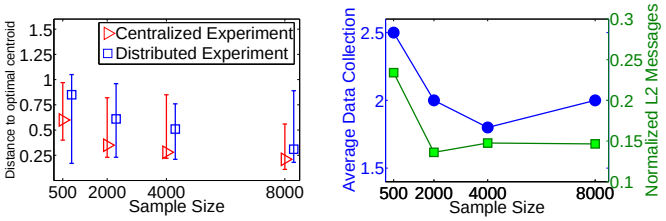
(a) Quality vs. ϵ (b) Cost vs. ϵ

Fig. 11. Dependency of cost and quality of mean monitoring on the alert threshold ϵ.

On the whole, quality of the mean monitoring algorithm outcome behaves well with respect to all the three parameters influencing it. The monitoring cost *i.e.* L2 messages is also low. Furthermore, on an average, the number of convergecast

rounds per epoch is around three – which can easily be reduced further by using a longer τ as the default value.

D. Experiments with k-Means Monitoring



(a) Average quality vs. sample size (b) Average monitoring cost vs. sample size

Fig. 12. Dependency of quality and cost of k-means monitoring on the sample size

In this set of experiments our goal is to investigate the effect of the sample size on the k-means monitoring algorithm. To do that we compare the results of our algorithm to those of a centralized algorithm that processed the entire data. We compute the distance between each centroid computed by the peer-to-peer algorithm and the closest centroid computed by the centralized one. Since our algorithm is not only distributed but also sample-based, we include for comparison the results of centralized algorithm which takes a sample from the entire data as its input. The most outstanding result, seen in Figure 12(a), is that most of the error of the distributed algorithm is due to sampling and not due to decentralization. The error, both average, best case, and worst case, is very similar to that of the centralized sample-based algorithm. This is significant in two ways. First, the decentralized algorithm is obviously an alternative to centralization; especially considering the far lower communication cost. Secondly, the error of the decentralized algorithm can be easily controlled by increasing the sample size.

The costs of k-means monitoring have to be separated to those related to monitoring the current centroids and those related to the collection of the sample. Figure 12(b) presents the costs of monitoring a single centroid and the number of times data was collected per epoch. These could be multiplied by k to bound the total costs (note that messages relating to different centroids can be piggybacked on each other). The cost of monitoring decreases drastically with increasing sample size – resulting from the better accuracy provided by the larger sample. Also there is a decrease in the number of convergecast rounds as the sample size increases. The default value of the alert mitigation factor τ in this experimental setup was 500. For any sample size greater than 2000, the number of convergecast rounds is about two per epoch – in the first round, it seems, the data is so much polluted by data from the previous epoch that a new round is immediately triggered. As noted earlier, this can be further decreased using a larger value of τ.

VII. Related Work

Algorithms for large distributed systems have been developed over the last half decade. These can be roughly classified into three categories: convergecast based or centralized algorithms, gossip based algorithms, and local algorithms. Some best-effort heuristics [11], [12], [13] were suggested as well.

The first category, convergecast based algorithms, is perhaps the simplest. Algorithms such as [14] provide generic solutions – suitable for the computation of multiple functions. They are also extremely communication efficient: computing the average, for instance, only requires one message from each peer. Some of these algorithms can be extremely synchronized – every round of computation taking a lot of time. This becomes very problematic when the data is dynamic and computation has to be iterated frequently. Other, such as STAR [15] can dynamically tune accuracy and timeliness vs. communication overhead. The most thorough implementation of this approach is possibly the Astrolabe system [16] which implement a general purpose infrastructure for distributed system monitoring.

The second category, gossip based algorithms, relies on the properties of random walks on graphs to provide probabilistic estimates for various statistics of data stored in the graph. Gossip based computation was first introduced by Kempe *et al.* [17], and have, since then, been expanded to general graphs by Boyd *et al.* [18]. The first gossip based algorithms required that the algorithm be executed from scratch if the data changes in order to maintain those guarantees. This problem was later addressed by Jelasity *et al.* [19]. The main benefit of our algorithm with respect to gossiping is that it is data driven. Thus, it is far more efficient than gossiping when the changes are stationary.

Local algorithms were first discussed by Afek *et al.* [20], Linial [21], and Naor and Stockmeyer [22], in the context of graph theory. Kutten and Peleg introduced local algorithms in which the input is data which is stored at the graph vertices, rather than the graph itself [23]. The first application of local algorithms to peer-to-peer data mining is the Majority-Rule algorithm by Wolff and Schuster [1]. Since then, local algorithms were developed for other data mining tasks *e.g.*, decision tree induction [24], multivariate regression [6], outlier detection [3], L2 norm monitoring [4], approximated sum [25], and more. The algorithm for L2 thresholding, and an initial application of that algorithm for k-means monitoring were first presented in a previous publication by the authors of this paper [4].

VIII. Conclusions and Open Questions

In this paper we present a generic algorithm which can compute *any* ordinal function of the average data in large distributed system. We present a number of interesting applications for this generic algorithm. Besides direct contributions to the calculation of L2 norm, the mean, and k-means in peer-to-peer networks, we also suggest a new reactive approach in which data mining models are computed by an approximate or heuristic method and are then efficiently judged by an efficient local algorithm.

This work leaves several interesting open questions. The first is the question of describing the "hardness" of locally computing a certain function $\mathcal{F}$ – its "locallability". For

instance, it is simple to show that majority voting lends itself better for local computation than the parity function. However, there is lack of an orderly method by which the hardness of these and other functions can be discussed. The second interesting question is the question of robustness of a generic local algorithm for general topologies. Last, in view of our generic algorithm it would be interesting to revisit Naor's and Stockmeyer's question [22] regarding the limitations of local computation.

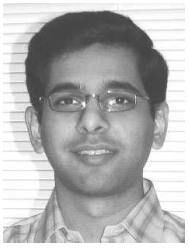
ACKNOWLEDGMENTS

This research is supported by the United States National Science Foundation CAREER award IIS-0093353 and NASA Grant NNX07AV70G.

REFERENCES

[1] R. Wolff and A. Schuster, "Association Rule Mining in Peer-to-Peer Systems," in *Proceedings of ICDM'03*, Melbourne, Florida, 2003, pp. 363–370.
[2] D. Krivitski, A. Schuster, and R. Wolff, "A Local Facility Location Algorithm for Sensor Networks," in *Proceedings of DCOSS'05*, Marina del Rey, California, 2005, pp. 368–375.
[3] J. Branch, B. Szymanski, R. Wolff, C. Gianella, and H. Kargupta, "In-Network Outlier Detection in Wireless Sensor Networks," in *Proceedings of ICDS'06*, Lisboa, Portugal, 2006, pp. 51–58.
[4] R. Wolff, K. Bhaduri, and H. Kargupta, "Local L2 Thresholding based Data Mining in Peer-to-Peer Systems," in *Proceedings of SDM'06*, Bethesda, Maryland, 2006, pp. 428–439.
[5] P. Luo, H. Xionga, K. Lu, and Z. Shi, "Distributed classification in peer-to-peer networks," in *Proceedings of SIGKDD'07*, San Jose, California, 2007, pp. 968–976.
[6] K. Bhaduri and H. Kargupta, "An Efficient Local Algorithm for Distributed Multivariate Regression in Peer-to-Peer Networks," in *Proceedings of SDM'08*, Atlanta, Georgia, 2008, pp. 153 – 164.
[7] N. Li, J. C. Hou, and L. Sha, "Design and Analysis of an MST-based Topology Control Algorithm," *IEEE Transactions on Wireless Communications*, vol. 4, no. 3, pp. 1195–1206, 2005.
[8] Y. Birk, L. Liss, A. Schuster, and R. Wolff, "A Local Algorithm for Ad Hoc Majority Voting Via Charge Fusion," in *Proceedings of DISC'04*, Amsterdam, Netherlands, 2004, pp. 275–289.
[9] K. Bhaduri, "Efficient Local Algorithms for Distributed Data Mining in Large Scale Peer to Peer Environments: A Deterministic Approach," Ph.D. dissertation, University of Maryland, Baltimore County, Baltimore, Maryland, USA, May 2008.
[10] K. Das, K. Bhaduri, K. Liu, and H. Kargupta, "Distributed Identification of Top-*l* Inner Product Elements and its Application in a Peer-to-Peer Network," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 20, no. 4, pp. 475–488, 2008.
[11] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, and S. Datta, "Clustering Distributed Data Streams in Peer-to-Peer Environments," *Information Science*, vol. 176, no. 14, pp. 1952–1985, 2006.
[12] W. Kowalczyk, M. Jelasity, and A. E. Eiben, "Towards Data Mining in Large and Fully Distributed Peer-to-Peer Overlay Networks," in *Proceedings of BNAIC'03*, Nijmegen, Netherlands, 2003, pp. 203–210.
[13] S. Datta, C. Giannella, and H. Kargupta, "K-Means Clustering over Large, Dynamic Networks," in *Proceedings of SDM'06*, Maryland, 2006, pp. 153–164.
[14] M. Rabbat and R. Nowak, "Distributed Optimization in Sensor Networks," in *Proceedings of IPSN'04*, California, 2004, pp. 20–27.
[15] N. Jain, D. Kit, P. Mahajan, P. Yalagandula, M. Dahlin, and Y. Zhang, "STAR: Self-tuning aggregation for scalable monitoring," in *Proceedings of VLDB'07*, Sept. 2007, pp. 962–973.
[16] R. van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 2, pp. 164–206, 2003.
[17] D. Kempe, A. Dobra, and J. Gehrke, "Computing Aggregate Information using Gossip," in *Proceedings of FOCS'03*, Cambridge, Massachusetts, 2003, pp. 482–491.
[18] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, " Gossip Algorithms: Design, Analysis and Applications," in *Proceedings of INFOCOM'05*, Miami, Florida, 2005, pp. 1653–1664.
[19] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based Aggregation in Large Dynamic Networks," *ACM Transactions on Computer Systems*, vol. 23, no. 3, pp. 219 – 252, 2005.
[20] Y. Afek, S. Kutten, and M. Yung, "Local Detection for Global Self Stabilization," *Theoretical Computer Science*, vol. 186, no. 1-2, pp. 199–230, 1997.
[21] N. Linial, "Locality in Distributed Graph Algorithms," *SIAM Journal of Computing*, vol. 21, no. 1, pp. 193–2010, 1992.
[22] M. Naor and L. Stockmeyer, "What can be Computed Locally?" in *Proceedings of STOC'93*, 1993, pp. 184–193.
[23] S. Kutten and D. Peleg, "Fault-Local Distributed Mending," in *Proceedings of PODC'95*, Ottawa, Canada, 1995, pp. 20–27.
[24] K. Bhaduri, R. Wolff, C. Giannella, and H. Kargupta, "Distributed Decision Tree Induction in Peer-to-Peer Systems," *Statistical Analysis and Data Mining Journal*, vol. 1, no. 2, pp. 85–103, 2008.
[25] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani, "The Price of Validity in Dynamic Networks," in *Proceedings of SIGMOD'04*, Paris, France, 2004, pp. 515–526.

Ran Wolff is faculty of the Management Information Systems department at University of Haifa, Israel. A graduate of the Technion – Israel, he previously held a post doctoral position at the University of Maryland in Baltimore County. His main fields of expertise are data mining in large-scale distributed environments: peer-to-peer networks, grid systems, and wireless sensor networks, and privacy preserving data mining. Ran regularly serves as PC in ICDM, SDM and SIGKDD, and as a reviewer for the DMKD and TKDE journals, among other. More information about him can be found at http://mis.haifa.ac.il/˜rwolff.



Kanishka Bhaduri received his B.E. in Computer Science and Engineering from Jadavpur University, India in 2003 and PhD degree in Computer Science from University of Maryland Baltimore County in 2008. Currently he is a research scientist with Mission Critical Technologies Inc at NASA Ames Research Center. His research interests include distributed and P2P data mining, data stream mining, and statistical data analysis. Kanishka serves as a reviewer for many conferences and journals such as ICDM, SDM, PKDD, SIGKDD, TKDE, TMC and more. More information about him can be found at http://www.csee.umbc.edu/˜kanishk1.

Hillol Kargupta is an Associate Professor at the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County. He received his Ph.D. in Computer Science from University of Illinois at Urbana-Champaign in 1996. He is also a co-founder of AGNIK LLC, a ubiquitous data intelligence company. His research interests include distributed data mining, data mining in ubiquitous environment, and privacy-preserving data mining. Dr. Kargupta won a US National Science Foundation CAREER award in 2001 for his research on ubiquitous and distributed data mining. He has published more than 90 peer-reviewed articles in journals, conferences, and books. He is an associate editor of the IEEE Transactions on Knowledge and Data Engineering, the IEEE Transactions on Systems, Man, and Cybernetics, Part B, and the Statistical Analysis and Data Mining Journal. He regularly serves on the organizing and program committees of many data mining conferences. More information about him can be found at http://www.csee.umbc.edu/˜hillol.